Tema 4: 📚 Guia Didático Completo: CRUD com SQLite em Java

Este guia explica **estruturas de controle** e **funções do CRUD** no código desenvolvido, destacando:

1. **Estruturas de Controle** (if, switch, while, try-catch, return)
2. **Funções do CRUD** (Create, Read, Update, Delete)
3. **Fluxo do Programa** (Menu interativo e conexão com banco de dados)

🔷 1. Estruturas de Controle no Código

1.1 Condição (if-else)

Usada para **tomar decisões** com base em condições.

Exemplo 1: Verificar se a conexão com o banco foi estabelecida

```
if (conn == null) {
   System.out.println("Erro ao conectar com o banco de dados.");
   return; // Encerra o programa se não houver conexão
}
```

Exemplo 2: Verificar se um produto existe antes de editá-lo

```
if (!produtoExiste(conn, id)) {
   System.out.println("Produto com ID " + id + " não encontrado.");
   return; // Sai da função se o produto não existir
}
```

1.2 Repetição (do-while e while)

Usada para **executar um bloco repetidamente** enquanto uma condição for verdadeira.

Exemplo 1: Menu principal (loop até o usuário sair)

```
do {
   System.out.println("\n=== Menu Principal ===");
   // ... opções do menu ...
   opcao = scanner.nextInt();

   switch (opcao) {
      // ... casos ...
   }
} while (opcao != 6); // Repete até o usuário escolher "Sair"
```

Exemplo 2: Listar produtos (loop no ResultSet)

```
while (rs.next()) { // Enquanto houver registros no banco
   System.out.println("ID: " + rs.getInt("id") +
            " | Nome: " + rs.getString("nome") +
            " | Preço: R$" + String.format("%.2f", rs.getDouble("preco")));
}
```

1.3 Seleção (switch-case)

Usada para **executar diferentes ações** com base no valor de uma variável.

Exemplo: Menu de opções

```
switch (opcao) {
    case 1:
        inserirProduto(conn, scanner);
        break;
    case 2:
        listarProdutos(conn);
        break;
    // ... outros casos ...
    default:
        System.out.println("Opção inválida. Tente novamente.");
}
```

1.4 Tratamento de Erros (try-catch)

Usado para **capturar e tratar exceções** (erros em tempo de execução).

Exemplo: Inserir um produto no banco

```
try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
    pstmt.setString(1, nome);
    pstmt.setDouble(2, preco);
    pstmt.executeUpdate();
    System.out.println("Produto inserido com sucesso!");
} catch (SQLException e) {
    System.out.println("Erro ao inserir produto: " + e.getMessage());
}
```

O que acontece?

- try: Tenta executar o código (inserir no banco).
- catch: Se ocorrer um erro (SQLException), exibe uma mensagem amigável.

1.5 Retorno de Funções (return)

Usado para **encerrar uma função** e, opcionalmente, retornar um valor.

Exemplo 1: Função que verifica se um produto existe

```
private static boolean produtoExiste(Connection conn, int id) {
    String sql = "SELECT 1 FROM produtos WHERE id = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        return rs.next(); // Retorna `true` se o produto existir
    } catch (SQLException e) {
        System.out.println("Erro ao verificar produto: " + e.getMessage());
        return false; // Retorna `false` em caso de erro
    }
}
```

Exemplo 2: Encerrando uma função prematuramente

```
if (!temAlteracao) {
    System.out.println("Nenhuma alteração foi informada.");
    return; // Sai da função sem fazer nada
}
```

🔷 2. Funções do CRUD (Create, Read, Update, Delete)

Operação	Função no Código	Descrição
Create (Inserir)	inserirProduto()	Adiciona um novo produto ao banco.
Read (Ler)	listarProdutos(), buscarProdutoPorId()	Lista todos os produtos ou busca um específico.
Update (Atualizar)	editarProduto()	Modifica nome ou preço de um produto existente.
Delete (Excluir)	excluirProduto()	Remove um produto do banco após confirmação.

2.1 Create: inserirProduto()

Objetivo: Inserir um novo produto no banco.

Passos:

1. Pede ao usuário o **nome** e **preço**.
2. Monta um comando SQL (INSERT).
3. Executa e verifica se foi inserido com sucesso.

Código:

```
public static void inserirProduto(Connection conn, Scanner scanner) {
    System.out.print("Nome do produto: ");
    scanner.nextLine(); // Limpa o buffer
    String nome = scanner.nextLine();
    System.out.print("Preço: ");
    double preco = scanner.nextDouble();

    String sql = "INSERT INTO produtos(nome, preco) VALUES(?, ?)";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, nome);
        pstmt.setDouble(2, preco);
        pstmt.executeUpdate();
        System.out.println("Produto inserido com sucesso!");
    } catch (SQLException e) {
        System.out.println("Erro ao inserir produto: " + e.getMessage());
    }
}
```

2.2 Read: listarProdutos() **e** buscarProdutoPorId()

Objetivo: Recuperar dados do banco.

listarProdutos()

Lista **todos** os produtos cadastrados.

```
public static void listarProdutos(Connection conn) {
```

```
    String sql = "SELECT * FROM produtos ORDER BY id";
    try (PreparedStatement pstmt = conn.prepareStatement(sql);
         ResultSet rs = pstmt.executeQuery()) {

        System.out.println("\n--- Lista de Produtos ---");
        while (rs.next()) {
            System.out.println("ID: " + rs.getInt("id") +
                               " | Nome: " + rs.getString("nome") +
                               " | Preço: R$" + String.format("%.2f", rs.getDouble("preco")));
        }
    } catch (SQLException e) {
        System.out.println("Erro ao listar produtos: " + e.getMessage());
    }
}
```

buscarProdutoPorId()

Busca **um produto específico** pelo ID.

```
public static void buscarProdutoPorId(Connection conn, Scanner scanner) {
    System.out.print("Digite o ID do produto: ");
    int id = scanner.nextInt();

    String sql = "SELECT * FROM produtos WHERE id = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            System.out.println("\n--- Produto Encontrado ---");
            System.out.println("ID: " + rs.getInt("id"));
            System.out.println("Nome: " + rs.getString("nome"));
            System.out.println("Preço: R$" + String.format("%.2f", rs.getDouble("preco")));
        } else {
            System.out.println("Produto com ID " + id + " não encontrado.");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao buscar produto: " + e.getMessage());
    }
}
```

2.3 Update: editarProduto()

Objetivo: Modificar um produto existente.

Passos:

1. Pede o **ID** do produto a ser editado.
2. Verifica se o produto existe.
3. Permite alterar **nome**, **preço** ou ambos.
4. Executa o UPDATE no banco.

Código:

```
public static void editarProduto(Connection conn, Scanner scanner) {
    System.out.print("Digite o ID do produto que deseja editar: ");
    int id = scanner.nextInt();

    if (!produtoExiste(conn, id)) {
        System.out.println("Produto com ID " + id + " não encontrado.");
        return;
    }

    System.out.print("Novo nome (deixe em branco para não alterar): ");
```

```
    scanner.nextLine();
    String novoNome = scanner.nextLine();

    System.out.print("Novo preço (digite 0 para não alterar): ");
    double novoPreco = scanner.nextDouble();

    // Monta o SQL dinamicamente
    StringBuilder sql = new StringBuilder("UPDATE produtos SET ");
    boolean temAlteracao = false;

    if (!novoNome.isEmpty()) {
        sql.append("nome = ?");
        temAlteracao = true;
    }

    if (novoPreco > 0) {
        if (temAlteracao) sql.append(", ");
        sql.append("preco = ?");
        temAlteracao = true;
    }

    if (!temAlteracao) {
        System.out.println("Nenhuma alteração foi informada.");
        return;
    }

    sql.append(" WHERE id = ?");

    try (PreparedStatement pstmt = conn.prepareStatement(sql.toString())) {
        int paramIndex = 1;

        if (!novoNome.isEmpty()) {
            pstmt.setString(paramIndex++, novoNome);
        }

        if (novoPreco > 0) {
            pstmt.setDouble(paramIndex++, novoPreco);
        }

        pstmt.setInt(paramIndex, id);

        int linhasAfetadas = pstmt.executeUpdate();

        if (linhasAfetadas > 0) {
            System.out.println("Produto atualizado com sucesso!");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao editar produto: " + e.getMessage());
    }
}
```

2.4 Delete: excluirProduto()

Objetivo: Remover um produto do banco.

Passos:

1. Pede o **ID** do produto a ser excluído.
2. **Confirma** a exclusão.
3. Executa o DELETE no banco.

Código:

```
public static void excluirProduto(Connection conn, Scanner scanner) {
    System.out.print("Digite o ID do produto que deseja excluir: ");
    int id = scanner.nextInt();
```

```
    if (!produtoExiste(conn, id)) {
        System.out.println("Produto com ID " + id + " não encontrado.");
        return;
    }

    System.out.print("Tem certeza que deseja excluir este produto? (S/N): ");
    scanner.nextLine();
    String confirmacao = scanner.nextLine().toUpperCase();

    if (!confirmacao.equals("S")) {
        System.out.println("Operação cancelada.");
        return;
    }

    String sql = "DELETE FROM produtos WHERE id = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        int linhasAfetadas = pstmt.executeUpdate();

        if (linhasAfetadas > 0) {
            System.out.println("Produto excluído com sucesso!");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao excluir produto: " + e.getMessage());
    }
}
```

🔷 3. Fluxo do Programa (Resumo)

1. **Conecta ao banco SQLite** (ConexaoSQLite.conectar()).
2. **Exibe um menu** com opções do CRUD.
3. **Executa a operação escolhida**:
    - Inserir, listar, editar, excluir ou buscar.
4. **Repete** até o usuário escolher "Sair".
5. **Fecha a conexão** ao final.

🎯 Conclusão

Este guia cobriu:
✓ **Estruturas de controle** (if, switch, while, try-catch, return).
✓ **Operações CRUD** (Create, Read, Update, Delete).
✓ **Fluxo do programa** e interação com o banco de dados.