Análise e Desenvolvimento de Sistemas
Programação Java
Profª. Ana Luiza Scharf
ana.scharf@ifsc.edu.br

Laboratório: Exemplo 03: Usando PreparedStatement

Objetivo

Demonstrar o uso seguro de comandos SQL em Java utilizando a interface `PreparedStatement`, evitando riscos como SQL Injection. Este exemplo utiliza a mesma estrutura de dados dos exemplos anteriores, mas com foco na segurança e boas práticas de consulta ao banco.

Pré-requisitos

Este exemplo é uma **continuação do Exemplo 01 e Exemplo2**, onde já foi criado o banco de dados `lab01.sqlite` e implementada uma classe básica de acesso ao banco usando JDBC.

Para prosseguir com este exemplo, é necessário:

- Ter concluído o Exemplo 01 e Exemplo02 com sucesso.
- Manter o arquivo `lab01.sqlite` no diretório raiz do projeto.

Organização das pastas esperada:

```
\begin{verbatim}
src/
 java/
    exemplo01/
       ExemploMuitoSimples.java
    exemplo02/
       PadroesDeProjeto.java
       db/
           ConnectionFactory.java
    exemplo03/
        UsandoPreparedStmt.java
    bcd/
        Principal.java
 lab01.sqlite
```

Crie a pasta `exemplo03` e dentro dela crie a classe `UsandoPreparedStmt`, responsável por fornecer conexões com o banco.

Classe `UsandoPreparedStmt`

Objetivo

Demonstrar o uso da classe `PreparedStatement` do Java para prevenir *SQL Injection* e facilitar a manipulação segura de dados em um banco SQLite.

Explicação do Código

Pacote e importações

```
package exemplo03;

import exemplo02.db.ConnectionFactory;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

O que faz:

- Define o pacote da classe.
- Importa recursos para conexão com banco de dados e execução de comandos SQL.

Declaração da classe e variável de formatação

```
public class UsandoPreparedStmt {
    private final String DIVISOR =
     "-------------------------------------------------------------------------------\n";
```

O que faz:

- Declara a classe principal do exemplo.
- Define uma string de separação para formatar melhor a saída.

Método: listarPessoas()

```
public String listarPessoas() throws SQLException {
    StringBuilder sb = new StringBuilder();
    String query = "SELECT * FROM Pessoa";

    try (Connection conexao = ConnectionFactory.getDBConnection();
         PreparedStatement stmt = conexao.prepareStatement(query);
         ResultSet rs = stmt.executeQuery()) {

        if (!rs.next()) {
            sb.append("\nNenhuma pessoa cadastrada no banco\n");
        } else {
            sb.append(DIVISOR);
            sb.append(String.format("|%-5s|%-25s|%-10s|%-10s|%-25s|\n", "ID", "Nome", "Peso", "Altura", "
     Email"));
            sb.append(DIVISOR);
            do {
                sb.append(String.format("|%-5d|%-25s|%-10.2f|%-10d|%-25s|\n",
                        rs.getInt("idPessoa"),
                        rs.getString("Nome"),
                        rs.getDouble("peso"),
                        rs.getInt("altura"),
                        rs.getString("email")));
            } while (rs.next());
            sb.append(DIVISOR);
        }
    } catch (SQLException e) {
        throw new SQLException(e);
    }
    return sb.toString();
}
```

O que faz:

- Consulta todos os registros da tabela `Pessoa`.
- Formata a saída de maneira organizada.
- Usa `PreparedStatement` para evitar SQL Injection.

Método: listarDadosDeUmaPessoa(int idPessoa)

```
public String listarDadosDeUmaPessoa(int idPessoa) throws SQLException {
    StringBuilder sb = new StringBuilder();
    String query = "SELECT * FROM Pessoa WHERE idPessoa = ?";

    try (Connection conexao = ConnectionFactory.getDBConnection();
         PreparedStatement stmt = conexao.prepareStatement(query)) {

        stmt.setInt(1, idPessoa);
        ResultSet rs = stmt.executeQuery();

        if (!rs.next()) {
            sb.append("\nNenhuma pessoa cadastrada no banco\n");
        } else {
            sb.append(DIVISOR);
            sb.append(String.format("|%-5s|%-25s|%-10s|%-10s|%-25s|\n", "ID", "Nome", "Peso", "Altura", "
     Email"));
            sb.append(DIVISOR);
            do {
                sb.append(String.format("|%-5d|%-25s|%-10.2f|%-10d|%-25s|\n",
                        rs.getInt("idPessoa"),
                        rs.getString("Nome"),
                        rs.getDouble("peso"),
                        rs.getInt("altura"),
                        rs.getString("email")));
            } while (rs.next());
            sb.append(DIVISOR);
        }
        rs.close();
    } catch (SQLException e) {
        throw new SQLException(e);
    }
    return sb.toString();
}
```

O que faz:

- Consulta os dados de uma única pessoa usando seu ID.
- Usa `?` como parâmetro para evitar SQL Injection.
- Formata a saída com cabeçalho e colunas alinhadas.

Método: atualizaEmail(int idPessoa, String email)

```
public int atualizaEmail(int idPessoa, String email) throws SQLException {
    int totalLinhasModificadas = 0;
    String query = "UPDATE Pessoa SET Email = ? WHERE idPessoa = ?";

    try (Connection conexao = ConnectionFactory.getDBConnection();
         PreparedStatement stmt = conexao.prepareStatement(query)) {

        stmt.setString(1, email);
        stmt.setInt(2, idPessoa);
        totalLinhasModificadas = stmt.executeUpdate();

    } catch (SQLException e) {
        throw new SQLException(e);
    }
    return totalLinhasModificadas;
}
```

O que faz:

- Atualiza o email de uma pessoa específica no banco de dados.
- Usa `PreparedStatement` com parâmetros para garantir segurança e flexibilidade.

Objetivo da classe `Principal`

A classe `Principal` foi criada no **Exemplo 01** para permitir que o usuário interaja com o sistema por meio de um menu simples no terminal. Neste exemplo, ela foi **modificada** para incluir uma nova opção no menu principal: executar o `Exemplo 03`, que permite o uso da classe responsável por interações seguras com o banco de dados usando `PreparedStatement`.

Modificações na classe `Principal`

```
private final String[] EXEMPLOS = {
    ...
    "3 - Exemplo 03 - uso de PreparedStatement",
    ...
};
```

O que faz:

- Adiciona a nova funcionalidade ao menu principal da aplicação.

Submenu do Exemplo 03

```
private final String[] MENU_EX3 = {
    "\n...:: Exemplo com PreparedStatement ::...\n",
    "1 - Listar todas pessoas",
    "2 - Listar dados de uma pessoa",
    "3 - Atualizar email de uma pessoa",
    "4 - Voltar ao menu anterior"
};
```

O que faz:

- Apresenta as opções disponíveis no contexto do uso de `PreparedStatement`.

Inclusão da lógica de execução

```
private void exemplo03() throws SQLException {
    int opcao;
    UsandoPreparedStmt app = new UsandoPreparedStmt();
    try {
        do {
            opcao = this.menu(this.MENU_EX3);
            switch (opcao) {
                case 1:
                    System.out.println(app.listarPessoas());
                    break;
                case 2:
                    System.out.print("Informe o ID da pessoa: ");
                    int idPessoa = teclado.nextInt();
                    System.out.println(app.listarDadosDeUmaPessoa(idPessoa));
                    break;
                case 3:
                    System.out.println(app.listarPessoas());
                    System.out.print("Informe o ID da pessoa que irá alterar o email: ");
                    idPessoa = teclado.nextInt();
                    System.out.print("Entre com o email: ");
                    String email = this.teclado.next();
                    if (app.atualizaEmail(idPessoa, email) > 0) {
                        System.out.println("Email atualizado com sucesso");
                    } else {
                        System.out.println("Não foi possível atualizar o email.");
                    }
                    break;
            }
        } while (opcao != 4);
    } catch (InputMismatchException e) {
        System.err.println("ERRO: Dados fornecidos estão em um formato diferente do esperado.");
```

```
    }
}
```

O que faz:

- Controla o menu e a execução do exemplo 03.
- Cria uma instância de `UsandoPreparedStmt` para chamar os métodos de acesso ao banco.
- Permite:
  - Listar todos os registros da tabela;
  - Consultar uma pessoa específica por ID;
  - Atualizar o email de uma pessoa de forma segura.

Chamada no `main()`

```
case 3:
    p.exemplo03();
    break;
```

O que faz:

- Executa a lógica da funcionalidade sempre que a opção 3 for escolhida no menu principal.

Como Executar a Classe `Principal` e o que Esperar

Para executar a classe `Principal`, siga os passos abaixo:

1. Execute a classe `bcd.Principal`. No terminal, o programa apresentará o menu principal com as opções:
   - **Exemplo 01:** permite gerenciar um cadastro simples de pessoas (inserir, alterar, excluir, listar);
   - **Exemplo 02:** demonstra o uso de padrões de projeto para acessar o banco de dados e listar pessoas;
   - **Exemplo 03:** demonstra o uso da interface `PreparedStatement` do JDBC para execução parametrizada e segura de comandos SQL;

Diferenças entre Exemplo 01, Exemplo 02 e Exemplo 03

- **Exemplo 01:** utiliza instruções SQL simples, com código acoplado diretamente ao método que executa a lógica de negócio.
- **Exemplo 02:** introduz o uso de padrões de projeto como `Factory`, separando a lógica de conexão em uma classe específica.
- **Exemplo 03:** utiliza `PreparedStatement`, uma forma segura de executar comandos SQL parametrizados, protegendo contra SQL Injection e facilitando a reutilização de comandos com diferentes parâmetros.