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

Laboratório: Exemplo 04 - Usando DAO (Detalhado)

Introdução

Neste laboratório, vamos aprender o padrão de projeto **DAO (Data Access Object)** que ajuda a organizar o acesso ao banco de dados de forma clara, segura e reutilizável.

Vamos analisar três classes principais:

- `Pessoa` - representa a entidade, ou seja, a tabela Pessoa no banco.
- `PessoaDAO` - responsável por acessar o banco e executar comandos SQL.
- `UsandoDAO` - mostra como usar o DAO para inserir e listar dados.

Organização das pastas esperada:

```
\begin{verbatim}
src/
 java/
    exemplo01/
       ExemploMuitoSimples.java
    exemplo02/
       PadroesDeProjeto.java
       db/
           ConnectionFactory.java
    exemplo03/
        UsandoPreparedStmt.java
    exemplo04/
       UsandoDAO.java
       entities/
           Pessoa.java
           PessoaDAO.java
    bcd/
        Principal.java
 lab01.sqlite
```

Classe `Pessoa` - Representando a Entidade

Código comentado

```
package exemplo04.entities;

/**
 * Classe que representa uma pessoa.
 * Cada atributo corresponde a uma coluna na tabela Pessoa.
 */
public class Pessoa {

    // Atributo que representa o identificador único da pessoa no banco
    private int idPessoa;

    // Nome da pessoa
    private String nome;

    // Peso da pessoa em kg
    private double peso;

    // Altura da pessoa em centímetros
    private int altura;

    // Email da pessoa
    private String email;

    // Construtor vazio necessário para frameworks e uso geral
    public Pessoa() { }

    // Construtor para inicializar a pessoa com dados
    public Pessoa(String nome, double peso, int altura, String email) {
        this.nome = nome;
        this.peso = peso;
        this.altura = altura;
        this.email = email;
    }

    // Getter para o idPessoa
    public int getIdPessoa() {
        return idPessoa;
    }

    // Setter para o idPessoa
    public void setIdPessoa(int idPessoa) {
        this.idPessoa = idPessoa;
    }

    // Getter para o nome
    public String getNome() {
        return nome;
    }

    // Setter para o nome
    public void setNome(String nome) {
        this.nome = nome;
    }

    // Getter para o peso
    public double getPeso() {
        return peso;
    }

    // Setter para o peso
    public void setPeso(double peso) {
        this.peso = peso;
    }

    // Getter para a altura
    public int getAltura() {
        return altura;
```

```
    }

    // Setter para a altura
    public void setAltura(int altura) {
        this.altura = altura;
    }

    // Getter para o email
    public String getEmail() {
        return email;
    }

    // Setter para o email
    public void setEmail(String email) {
        this.email = email;
    }

    /**
     * Método que retorna uma string formatada da pessoa,
     * muito útil para exibir os dados organizadamente no console.
     */
    @Override
    public String toString() {
        return String.format("|%-5d|%-25s|%-10.2f|%-10d|%-25s|",
                idPessoa, nome, peso, altura, email);
    }
}
```

Explicação passo a passo

- **package exemplo04.entities;**
  Define o pacote onde a classe está. Isso ajuda a organizar seu projeto e evitar conflitos.
- **Atributos privados**
  Cada atributo representa uma coluna da tabela `Pessoa`. Eles são privados para proteger os dados e forçar o uso dos métodos getters e setters.
- **Construtores**
  - O construtor vazio é importante para permitir que frameworks (e.g. JDBC, JPA) possam instanciar objetos.
  - O construtor completo facilita criar objetos preenchidos rapidamente.
- **Getters e Setters**
  Métodos públicos para acessar (get) e modificar (set) os atributos privados.
- **toString()**
  Sobrescreve o método padrão para formatar os dados da pessoa de modo que facilite a leitura, especialmente quando queremos mostrar os dados no console.

Classe `PessoaDAO` - Data Access Object

Código comentado

```
package exemplo04.entities;

import exemplo02.db.ConnectionFactory;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

/**
 * Essa classe é responsável por acessar a tabela Pessoa no banco de dados.
 * Centraliza os comandos SQL e evita que eles fiquem espalhados no projeto.
 */
public abstract class PessoaDAO {

    /**
     * Método para adicionar uma nova pessoa no banco.
     * @param p Objeto Pessoa com os dados a inserir.
     * @return true se a operação foi bem sucedida.
     * @throws SQLException em caso de erro na conexão ou comando SQL.
     */
    public final static boolean adiciona(Pessoa p) throws SQLException {
        boolean resultado = false;

        // Comando SQL para inserir os dados na tabela Pessoa
        String sql = "INSERT INTO Pessoa (nome, peso, altura, email) VALUES (?,?,?,?)";

        // Try-with-resources para garantir que a conexão e statement sejam fechados automaticamente
        try (Connection conexao = ConnectionFactory.getDBConnection();
             PreparedStatement stmt = conexao.prepareStatement(sql)) {

            // Seta os valores para os parâmetros ? na ordem correta
            stmt.setString(1, p.getNome());
            stmt.setDouble(2, p.getPeso());
            stmt.setInt(3, p.getAltura());
            stmt.setString(4, p.getEmail());

            // Executa o comando SQL
            resultado = stmt.execute();
        } catch (SQLException ex) {
            // Relança a exceção para ser tratada onde o método for chamado
            throw new SQLException(ex);
        }
        return resultado;
    }

    /**
     * Método para listar todas as pessoas cadastradas no banco.
     * @return Uma lista de objetos Pessoa.
     * @throws SQLException em caso de erro na consulta.
     */
    public final static List<Pessoa> listarTodas() throws SQLException {
        List<Pessoa> pessoas = new ArrayList<>();

        // Comando SQL para selecionar todos os registros da tabela Pessoa
        String sql = "SELECT * FROM Pessoa";

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

            // Enquanto houver registro no resultado, cria objeto Pessoa e adiciona à lista
            while (rs.next()) {
                Pessoa c = new Pessoa(
                        rs.getString("nome"),
                        rs.getDouble("peso"),
```

```
                        rs.getInt("altura"),
                        rs.getString("email"));
                c.setIdPessoa(rs.getInt("idPessoa"));
                pessoas.add(c);
            }
        } catch (SQLException ex) {
            throw new SQLException(ex);
        }

        return pessoas;
    }
}
```

Explicação detalhada

- **Importações:**
  Importa as classes necessárias para conexão com banco, execução de comandos SQL e manipulação de resultados.

- **Classe abstrata:**
  Definida como `abstract` porque não queremos instanciar objetos dela, mas usar seus métodos estáticos.

- **Método `adiciona()`:**
  - Recebe um objeto `Pessoa` para inserir no banco.
  - Define a string SQL com parâmetros `?` para evitar SQL Injection.
  - Usa `PreparedStatement` para preencher esses parâmetros de forma segura.
  - Executa o comando e retorna o resultado.
  - Usa try-with-resources para garantir que conexões sejam fechadas automaticamente, evitando vazamento de recursos.

- **Método `listarTodas()`:**
  - Executa um `SELECT *` para pegar todas as pessoas cadastradas.
  - Percorre o `ResultSet` com `while(rs.next())` para criar objetos `Pessoa` com os dados do banco.
  - Adiciona cada objeto criado a uma lista que é retornada no final.
  - Também usa try-with-resources para segurança e limpeza dos recursos.

Classe `UsandoDAO` - Exemplo de Uso do DAO

Código comentado

```
package exemplo04;

import exemplo04.entities.Pessoa;
import exemplo04.entities.PessoaDAO;

import java.sql.SQLException;
import java.util.List;

/**
 * Esta classe demonstra como usar os métodos da classe PessoaDAO
 * para acessar os dados de forma simples e organizada.
 */
public class UsandoDAO {
    // String para formatar visualmente a saída
    private final String DIVISOR =
     "-----------------------------------------------------------------------------\n";

    /**
     * Método para cadastrar uma pessoa no banco, usando PessoaDAO.
     * @param p Objeto Pessoa com os dados a inserir.
     * @return true se o cadastro foi realizado com sucesso.
     * @throws SQLException em caso de erro ao acessar o banco.
     */
    public boolean cadastrarPessoa(Pessoa p) throws SQLException {
        // Delegamos a responsabilidade de inserir ao PessoaDAO
        return PessoaDAO.adiciona(p);
    }

    /**
     * Método para listar todas as pessoas cadastradas, formatando a saída.
     * @return String formatada com os dados de todas as pessoas.
     * @throws SQLException em caso de erro na consulta.
     */
    public String listarPessoas() throws SQLException {
        // Busca a lista de pessoas pelo DAO
        List<Pessoa> pessoas = PessoaDAO.listarTodas();

        StringBuilder sb = new StringBuilder();

        // Cabeçalho da tabela
        sb.append(DIVISOR);
        sb.append(String.format("|%-5s|%-25s|%-10s|%-10s|%-25s|\n", "ID", "Nome", "Peso", "Altura", "Email"))
     ;
        sb.append(DIVISOR);

        // Percorre a lista usando lambda para adicionar cada pessoa formatada na StringBuilder
        pessoas.forEach(pessoa -> sb.append(pessoa.toString()).append("\n"));

        sb.append(DIVISOR);

        // Retorna a string com os dados organizados
        return sb.toString();
    }
}
```

Explicação passo a passo

- **Importações:**
  Importa as classes `Pessoa` e `PessoaDAO` para manipular os dados e o acesso ao banco.
- **Atributo `DIVISOR`:**
  Uma linha longa para separar visualmente a tabela no terminal, melhorando a leitura.
- **Método `cadastrarPessoa()`:**

Recebe um objeto `Pessoa` e chama o método `adiciona()` da classe `PessoaDAO` para inserir no banco. Essa separação deixa o código mais organizado e reaproveitável.

- **Método `listarPessoas()`:**
  - Solicita ao DAO a lista de pessoas cadastradas.
  - Usa um `StringBuilder` para montar a saída formatada.
  - Adiciona um cabeçalho com os nomes das colunas e o divisor para organizar a visualização.
  - Usa uma expressão lambda para percorrer a lista e adicionar cada pessoa formatada.
  - Retorna a string completa para ser exibida no console ou outro destino.

Classe `Principal` - Integração do Exemplo 04 (DAO)

Trecho descomentado do menu principal

```
// No vetor EXEMPLOS, ativamos a opção 4 para o exemplo 04
private final String[] EXEMPLOS = {
    "\n..:: Pequenos exemplos com Java, SQLite e MySQL ::..\n",
    "1 - Exemplo 01",
    "2 - Exemplo 02 - uso de padrões de projeto",
    "3 - Exemplo 03 - uso de PreparedStatement",
    "4 - Exemplo 04 - uso do Data Access Object (DAO)",
    //"5 - Exemplo 05 - MySQL",
    "6 - Sair do programa"
};
```

Explicação: Aqui estamos incluindo no menu principal a opção 4, que permite executar as funcionalidades do exemplo 04, que é o uso do padrão DAO. Assim o usuário pode escolher essa opção para acessar as funcionalidades desse exemplo.

—

Descomentando o menu específico do exemplo 04

```
private final String[] MENU_EX4 = {
    "\n...:: Exemplo com Data Access Object (DAO) ::...\n",
    "1 - Cadastrar pessoa",
    "2 - Listar todas pessoas",
    "3 - Voltar ao menu anterior"
};
```

Explicação: Este vetor é o submenu específico para o exemplo 04, apresentando as opções para o usuário cadastrar uma pessoa, listar todas ou voltar para o menu anterior.

—

Método `exemplo04()` descomentado e explicado

```
private void exemplo04() throws SQLException {
    int opcao;
    UsandoDAO app = new UsandoDAO();  // Instância do DAO para manipular dados
    try {
        do {
            opcao = this.menu(this.MENU_EX4); // Exibe o menu específico do exemplo 04

            switch (opcao) {
                case 1:  // Cadastrar pessoa
                    try {
                        teclado.nextLine(); // Limpa o buffer do teclado

                        // Entrada de dados do usuário
                        System.out.print("Entre com o nome: ");
                        String nome = teclado.nextLine();

                        System.out.print("Entre com o email: ");
                        String email = teclado.nextLine();

                        System.out.print("Entre com o peso: ");
                        double peso = teclado.nextDouble();

                        System.out.print("Entre com a altura: ");
                        int altura = teclado.nextInt();

                        // Criação do objeto Pessoa com os dados informados
                        Pessoa p = new Pessoa(nome, peso, altura, email);

                        // Chamada do método para cadastrar no banco via DAO
                        boolean resultado = app.cadastrarPessoa(p);
```

```
                        if (resultado) {
                            System.out.println("\nPessoa cadastrada com sucesso.\n");
                        } else {
                            System.out.println("\nHouve algum problema e não foi possível cadastrar");
                        }
                    } catch (Exception e) {
                        System.err.println("\nErro com os dados fornecidos. Tente novamente.\n");
                    }
                    break;

                case 2:  // Listar todas as pessoas cadastradas
                    System.out.println(app.listarPessoas());
                    break;
            }
        } while (opcao != 3); // Voltar ao menu anterior
    } catch (InputMismatchException e) {
        System.err.println("ERRO: Dados fornecidos estão em um formato diferente do esperado.");
    }
}
```

Explicação detalhada:

- Criamos uma instância de `UsandoDAO` para usar seus métodos de cadastro e listagem.
- O menu é exibido repetidamente até que o usuário escolha a opção 3 para sair.
- Opção 1:
  - Limpa o buffer do teclado para evitar erros de leitura.
  - Pede para o usuário informar os dados da pessoa (nome, email, peso, altura).
  - Cria um objeto `Pessoa` com esses dados.
  - Chama o método `cadastrarPessoa()` que usa o DAO para inserir no banco.
  - Informa ao usuário se o cadastro foi bem sucedido ou se houve erro.
- Opção 2:
  - Chama o método `listarPessoas()` do DAO e imprime o resultado formatado no console.
- Exceções de formato errado no teclado são capturadas e avisam o usuário para corrigir.

—

Alteração no `main()` para incluir o exemplo 04

```
public static void main(String[] args) throws Exception {
    Principal p = new Principal();
    int opcao = -1;
    do {
        opcao = p.menu(p.EXEMPLOS);
        switch (opcao) {
            case 1:
                p.exemplo01();
                break;
            case 2:
                p.exemplo02();
                break;
            case 3:
                p.exemplo03();
                break;
            case 4:   // Adicionada chamada ao exemplo04
                p.exemplo04();
                break;
            // case 5:
            //     p.exemplo05();
            //     break;
        }
    } while (opcao != 6);
}
```

Explicação: Agora, ao escolher a opção 4 no menu principal, o programa chama o método `exemplo04()` para o usuário interagir com o DAO, podendo cadastrar e listar pessoas.

Resumo final

Este exemplo mostra como o padrão DAO ajuda a manter seu código limpo e modularizando o acesso a dados. Você não precisa repetir comandos SQL espalhados no seu projeto, apenas chama os métodos do DAO.

Assim, se você precisar mudar a forma de acesso (por exemplo, mudar o banco ou a consulta), altera apenas o DAO, e o resto do sistema continua funcionando normalmente.

Este projeto foi integralmente elaborado pelo professor **Emerson Ribeiro de Mello**, docente do IFSC – Campus São José.