

Desenvolvimento Orientado à Objetos

Persistência - JDBC

Christien Lana Rachid
2019/1

JDBC | História | ODBC

- ❑ Cada SGBD possui necessidades específicas.
 - ❑ Interface para comunicação diferente.
- ❑ Em 1992, foi proposto o ODBC (**O**pen **D**atabase **C**onnectivity)
 - ❑ Discutido e proposto pelo SQL Access Group (SAG)
 - ❑ Encabeçado pela Microsoft
 - ❑ Padrão para se conectar a qualquer banco de dados da mesma forma.
 - ❑ Dependendo apenas da existência de um driver no SO.

JDBC | História | ODBC

- ❑ ODBC é dependente de um driver específico do Sistema Operacional.
 - ❑ “Ponte” para implementação Nativa
 - ❑ Dependente de plataforma.

JDBC | História | JDBC

- ❑ Para evitar essa dependência a Sun Microsystems criou o JDBC.
- ❑ Escrito em Java
- ❑ Independente de plataforma
- ❑ Parte do SDK padrão do Java
- ❑ É possível conectar com o driver ODBC

JDBC | Principais Estruturas | Registrando a classe

- ❑ Registrar a classe referente ao driver JDBC.
- ❑ Garante que a classe estará disponível para o DriverManager.

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

JDBC | Principais Estruturas | Criando Conexão

- ❑ Utilizar o **DriverManager** para acessar o driver do banco de dados específico e se conectar ao banco de dados.
- ❑ Detalhes específicos de conexão são definidos por meio da URL de comunicação.
 - ❑ Detalhes dessa URL podem variar de acordo com o driver JDBC.

```
String url = "jdbc:mysql://localhost/ces";  
Connection con = DriverManager.getConnection(url, "<USR>", "<PAS>");
```

JDBC | Principais Estruturas | Criando Conexão

- ❑ Utilizar o **DriverManager** para acessar o driver do banco de dados específico e se conectar ao banco de dados.
- ❑ Detalhes específicos de conexão são definidos pelo por meio da URL de comunicação.
 - ❑ Detalhes dessa URL podem variar de acordo com o driver JDBC.

```
String url = "jdbc:mysql://HOST:databaseName=<BANCO>";  
Connection con = DriverManager.getConnection(url, "<USR>", "<PAS>");
```

JDBC | Principais Estruturas | Preparando Statement

- ❑ Um **Statement** corresponde a um “*comando*” no banco de dados.
- ❑ Existem vários tipos especiais de Statement, para comandos de alteração e chamadas de função.

```
Statement stmt = con.createStatement();
```


JDBC | Principais Estruturas | Realizando consulta

- ❑ Um **Statement** permite a realização de uma consulta, em formato de String.
- ❑ Essa operação retorna um **ResultSet** em caso de sucesso ou dispara uma exceção em caso de falha.

```
ResultSet rs = stmt.executeQuery(query);
```

JDBC | Principais Estruturas | Manipulando resultado

- ❑ Um **ResultSet** permite que os resultados sejam lidos linha a linha.

```
while (rs.next())  
{  
    // Obtendo o campo name em um string  
    String s = rs.getString("name");  
    // Obtendo o campo id em um inteiro  
    int i = rs.getInt("id");  
    System.out.println(s + " " + i);  
}
```

JDBC | Principais Estruturas | Fechando recursos

- ❑ Atenção! Conexões e Statements consomem recursos, lembre-se de encerrá-los.

```
stmt.close();  
rs.close();
```

JDBC | Principais Estruturas | Fechando recursos

- ❑ Atenção! Conexões e Statements consomem recursos, lembre-se de encerrá-los.

```
stmt.close()  
con.close()
```

JDBC | Exemplo 1

```
package jdbc;

import java.sql.*;

public class APPPersistencia01 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try{
            JOptionPane.showMessageDialog(null, "Testando a conexão.");
            Connection con = new ConectionFactory().conecta();
            // JOptionPane.showMessageDialog(null, "Conexão realizada com sucesso.");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("select * from uf");
            while (rs.next()) {
                int id = rs.getInt("iduf");
                String nome = rs.getString("sigla");
                String r = "ID:"+id + "-" + "UF:" + nome;
                System.out.println(r);
            }

        }catch(Exception e){
            JOptionPane.showMessageDialog(null, "Erro: "+e);
        }
    }
}
```









JDBC | Conectando em outros bancos!

- ❑ É necessário disponibilizar na pasta **lib** o **.jar** com o driver JDBC específico do banco de dados desejado.











SGBD	Driver	URL
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://HOST/DATABASE
SQL Server	com.microsoft.jdbc.sqlserver.SQLServerDriver	jdbc:microsoft:sqlserver://HOST:1433;DatabaseName=DATABASE
Postgre	org.postgresql.Driver	jdbc:postgresql://HOST/DATABASE
Oracle	oracle.jdbc.OracleDriver	jdbc:oracle:thin:@//HOST/DATABASE

JDBC | Atividade

 Crie um banco de dados **academia** com uma tabela **aluno**, contendo os seguintes campos:

-  **id**, inteiro, auto incremento, chave primária
-  **nome**, alfanumérico de tamanho 45
-  **dataMatricula**, data
-  **endereco**, TEXT
-  **telefone**, alfanumérico de tamanho 15
-  **dataNascimento**, data
-  **altura**, ponto flutuante
-  **peso**, ponto flutuante

 Solução

```
 CREATE TABLE aluno (  
     id INT IDENTITY(1,1) PRIMARY KEY,  
     nome VARCHAR(45),  
     data_matricula DATE,  
     endereco TEXT,  
     telefone VARCHAR(15),  
     data_nascimento DATE,  
     altura NUMERIC(15,2),  
     peso NUMERIC(15,2)  
 )
```

JDBC | Atividade

- ❑ Insira na tabela criada, 5 alunos.
- ❑ Escreva um programa, utilizando JDBC, que imprima todos oos alunos no seguinte formato:

```
| ID | NOME | DATA MATRICULA| DATA NASC. | TELEFONE      | PESO | ALTURA | ENDERECO  
| 1  | José | 01/02/2017      | 01/08/1990 | 32 988544561 | 90,0 | 180,0 | Rua José Louren...
```

```
insert into aluno (nome, data_matricula, endereco, telefone, data_nascimento, altura, peso)  
values ('Aluno 2', '2016-12-01', 'Rua José Lourenço kelmer', '32949995522', '2017-01-01', 1.72, 109.5);  
insert into aluno (nome, data_matricula, endereco, telefone, data_nascimento, altura, peso)  
values ('Aluno 3', '2016-12-01', 'Rua X', '32949995522', '2017-01-01', 1.72, 109.5);  
insert into aluno (nome, data_matricula, endereco, telefone, data_nascimento, altura, peso)  
values ('Aluno 4', '2016-12-01', 'Rua Y', '32949995522', '2017-01-01', 1.72, 109.5);  
insert into aluno (nome, data_matricula, endereco, telefone, data_nascimento, altura, peso)  
values ('Aluno 5', '2016-12-01', 'Z', '32949995522', '2017-01-01', 1.72, 109.5);  
insert into aluno (nome, data_matricula, endereco, telefone, data_nascimento, altura, peso)  
values ('Aluno 6', '2016-12-01', 'W', '32949995522', '2017-01-01', 1.72, 109.5);
```


JDBC | Atividade 2

❑ Crie um objeto Java que corresponda à tabela criada, incluindo atributos para cada campo, getters e setters e uma implementação de `toString()`, um construtor com todos os atributos e um construtor vazio.

- ❑ **id**, Long
- ❑ **nome**, String
- ❑ **dataMatricula**, Date
- ❑ **endereco**, String
- ❑ **telefone**, Date
- ❑ **dataNascimento**, Date
- ❑ **altura**, Float
- ❑ **peso**, Float

JDBC | CRUD | Inserção | Criando um PreparedStatement

- ❑ Operação de cadastro de dados no banco de dados
 - ❑ (C - Create)
- ❑ Para operações no banco de dados baseadas em parâmetros, utilize sempre PreparedStatement
 - ❑ Implementação otimizada para criação dos SQL.
 - ❑ Evita concatenações desnecessárias.
 - ❑ É uma forma segura de se evitar **SQL Injection**
- ❑ Em um ambiente Orientado à Objetos, criamos inserções a partir de objetos.
 - ❑ Mapeamento objeto-relacional

JDBC | CRUD | Inserção | Criando um PreparedStatement

// Criação de datas específicas em Java ;)

```
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
```

```
java.util.Date dataNasc = sdf.parse("01/08/1990");
```

// 1 - Instancie o seu Aluno!

```
Aluno a = new Aluno ( .....);
```

// 2 - Utilize o PreparedStatement para preparar a inserção.

```
String sql = "INSERT INTO aluno (nome, dataMatricula, endereco, telefone, dataNascimento, altura, peso) " +  
            "VALUES (?, ?, ?, ?, ?, ?, ?)";
```

// 3 - Defina os parâmetros para a inserção com base no objeto.

```
PreparedStatement statement = conn.prepareStatement(sql);
```

```
statement.setString(1, a.getNome());
```

```
statement.setDate(2, a.getDataMatricula());
```

```
statement.setString(3, a.getEndereco());
```

```
statement.setString(4, a.getTelefone());
```

```
statement.setDate(5, a.getDataNascimento());
```

```
statement.setFloat(6, a.getAltura());
```

```
statement.setFloat(7, a.getpeso());
```

// 4 - Aplica a operação no banco de dados.

```
int rowsInserted = statement.executeUpdate();
```

```
if (rowsInserted > 0) {
```

```
    System.out.println("Aluno inserido com sucesso!");
```

```
}
```

JDBC | CRUD | Seleção

- ❑ Operação de recuperação de dados do Banco de Dados
 - ❑ (R - Retrieval)
- ❑ Em um ambiente Orientado à Objetos, precisamos obter os dados e construir representações dos mesmos em forma de objetos.
 - ❑ Mapeamento Objeto Relacional

JDBC | CRUD | Inserção | Do ResultSet para Objetos

```
// 1 Prepara Lista de Alunos
```

```
List<Aluno> alunos = new ArrayList<>();
```

```
// 2 Prepara SQL para obter os dados de aluno
```

```
String sql = "SELECT id, nome, dataMatricula, endereco, telefone, dataNascimento, altura, peso FROM aluno";
```

```
// 3 Executa a consulta SQL
```

```
Statement statement = conn.createStatement();
```

```
ResultSet result = statement.executeQuery(sql);
```

```
// 4 Enquanto houver mais resultados
```

```
while (result.next()){
```

```
    // 5 Cria um novo aluno
```

```
    Aluno aluno = new Aluno();
```

```
    // 6 Preenche os dados do aluno
```

```
    aluno.setId(result.getLong("id"));
```

```
    aluno.setNome(result.getString("nome"));
```

```
    aluno.setDataMatricula(result.getDate("dataMatricula"));
```

```
    aluno.setEndereco(result.getString("endereco"));
```

```
    aluno.setTelefone(result.getString("telefone"));
```

```
    aluno.setDataNascimento(result.getDate("dataNascimento"));
```

```
    aluno.setAltura(result.getFloat("altura"));
```

```
    aluno.setPeso(result.getFloat("peso"));
```

```
    // 7 Adiciona o aluno na lista de alunos
```

```
    alunos.add(aluno)
```

```
}
```

JDBC | Atividade

- ❑ Utilizando os conceitos de inserção e consulta de dados com JDBC:
 - ❑ Crie um método **insere(Aluno aluno)** que recebe um aluno e adiciona esse aluno no banco.
 - ❑ Crie um método **List<Aluno> listar()** que lista todos os alunos cadastrados no banco.
 - ❑ No método **main**, crie 5 objetos **Aluno** e utilizando os métodos acima, insira cada um deles e depois liste do banco o resultado.

JDBC | CRUD | Atualização

- ❑ Operação de atualização de dados existentes no banco de dados (U - Update)
- ❑ Para operações no banco de dados baseadas em parâmetros, utilize sempre PreparedStatement
 - ❑ Implementação otimizada para criação dos SQL.
 - ❑ Evita concatenações desnecessárias.
 - ❑ É uma forma segura de se evitar **SQL Injection**
- ❑ Em um ambiente Orientado à Objetos, criamos atualizações a partir de objetos.

JDBC | CRUD | Atualização | Do ResultSet para Objetos

```
Aluno aluno = new Aluno(.....);
```

```
// 1 Prepara SQL para obter os dados de aluno
```

```
String sql = "UPDATE aluno SET nome=?, dataMatricula=?, endereco=?, telefone=?, dataNascimento=?, altura=?,  
peso=? WHERE id=?";
```

```
// 2 - Defina os parâmetros para a atualização com base no objeto.
```

```
PreparedStatement statement = conn.prepareStatement(sql);
```

```
statement.setString(1, a.getNome());
```

```
statement.setDate(2, a.getDataMatricula());
```

```
statement.setString(3, a.getEndereco());
```

```
statement.setString(4, a.getTelefone());
```

```
statement.setDate(5, a.getDataNascimento());
```

```
statement.setFloat(6, a.getAltura());
```

```
statement.setFloat(7, a.getpeso());
```

```
statement.setString(8, a.getId());
```

```
// 3 - Aplica a operação no banco de dados.
```

```
int rowsInserted = statement.executeUpdate();
```

```
if (rowsInserted > 0) {
```

```
    System.out.println("Aluno atualizado com sucesso!!");
```

```
}
```


JDBC | CRUD | Remoção

- ❑ Operação de remove dados existentes no banco de dados (D - Delete)
- ❑ Para operações no banco de dados baseadas em parâmetros, utilize sempre PreparedStatement
- ❑ Em um ambiente Orientado à Objetos, criamos remoções a partir de objetos.

JDBC | CRUD | Inserção | Do ResultSet para Objetos

```
Aluno aluno = new Aluno(.....);
```

```
// 1 Prepara SQL para obter os dados de aluno
```

```
String sql = "DELETE FROM aluno WHERE id = ?";
```

```
// 2 - Defina os parâmetros para a remoção com base no objeto.
```

```
PreparedStatement statement = conn.prepareStatement(sql);
```

```
statement.setString(1, a.getId());
```

```
// 3 - Aplica a operação no banco de dados.
```

```
int rowsInserted = statement.executeUpdate();
```

```
if (rowsInserted > 0) {
```

```
    System.out.println("Aluno removido com sucesso!!");
```

```
}
```

JDBC | Atividade 2

- ❑ Utilizando os conceitos de atualização e deleção de dados com JDBC:
 - ❑ Crie um método **atualiza(Aluno aluno)** que recebe um aluno e atualiza os dados do aluno no banco com base em seu ID.
 - ❑ Crie um método **remove(Aluno aluno)** que remove um aluno do banco de dados com base em seu ID
- ❑ No método **main**, liste todos os usuários do banco utilizando o método **Listar**, atualize metade deles para o peso 90 quilos e a outra metade para o peso 100 quilos utilizando o método **atualizar**. Em seguida, utilize o método **remover** para apagar todos os que possuem **ids** ímpares.

Código da parte 1: <http://tinyurl.com/jdbccrud>