

JDBC (Java Database Connectivity)

Padrão de Projeto DAO (Data Access Object)

Roteiro para "instalação" do banco de dados e do driver JDBC

# Persistência em Java

---

# JDBC

- Java Database Connectivity
- Provê acesso SQL ao banco de dados
- Principais componentes:
  - DriverManager
  - Connection
  - Statement
  - PreparedStatement
  - ResultSet
  - SQLException

# DriverManager

- Usa um driver **disponível** para realizar uma conexão com o banco de dados
- Método estático **getConnection** que retorna uma **Connection** e recebe os seguintes parâmetros:
  - Localização do **host**, **porta** e **schema**
  - Usuário
  - Senha

# Connection

- **Representa** uma conexão com o banco de dados e pode ser usada para **criar** um **Statement** ou para **preparar** um **PreparedStatement**, por meio, respectivamente dos métodos:
  - `createStatement()`
  - `prepareStatement(sql)`

# Criação da conexão

```
String driver = "org.gjt.mm.mysql.Driver";
String strConexao =
    "jdbc:mysql://localhost/persistenciaUmaTabela";
String usuario = "root";
String senha = "";
try {
    Class.forName(driver);
    Connection conn =
        DriverManager.getConnection(
            strConexao, usuario, senha);
} catch (ClassNotFoundException ex) {
    System.out.println(
        "Driver do banco nao encontrado.");
}
```

# Statement

- **Representa** um instrução **SEM** parâmetros, a ser enviada para o banco de dados, pelo acionamento dos métodos:
  - **int executeUpdate()**, para instruções **INSERT**, **UPDATE** e **DELETE**
  - **ResultSet executeQuery()**, para instruções **SELECT**

# PreparedStatement

- **Representa** um instrução **COM** parâmetros a ser enviada para o banco de dados pelo acionamento dos métodos:
  - `int executeUpdate()`, para instruções INSERT, UPDATE e DALETE
  - `ResultSet executeQuery()`, para instruções SELECT
- **Antes** da execução dos métodos acima, os **parâmetros** devem ser **configurados** com os métodos **`setString()`**, **`setInt()`**, etc

# ResultSet

- Conjunto de **registros** retornados depois da execução de **Statement.executeQuery()** ou **PreparedStatement.executeQuery()**
- Principais métodos:
  - **booleana next()**: avança e cursor e indica se há dados
  - **int getInt(String)** e **String getString(String)**: retorna o dado de uma coluna com um nome especificado.

# SQLException

- **Representa** uma **exceção** retornada pela API JDBC
- **Deve** ser tratada (**try-catch**) ou declarada para ser passada (**throws**) ao método chamador

# Leitura com Statement

```
try {  
  
    Statement stmt = conn.createStatement();  
    ResultSet rs =  
        stmt.executeQuery("SELECT * FROM empregado");  
  
    while (rs.next()) {  
        System.out.println(rs.getString("codigo"));  
        System.out.println(rs.getString("nome"));  
        System.out.println(rs.getInt("idade"));  
    }  
  
} catch (SQLException e) {  
    // tratamento da exceção  
}
```

# Leitura com PreparedStatement

```
try {
    PreparedStatement stmt =
        conn.createStatement(
            "SELECT * FROM empregado WHERE codigo = ?");
    stmt.setString(1, codigoDePesquisa);
    ResultSet rs = stmt.executeQuery();

    if (rs.next()) {
        System.out.println(rs.getString("codigo"));
        System.out.println(rs.getString("nome"));
        System.out.println(rs.getInt("idade"));
    }
} catch (SQLException e) {
    // tratamento da exceção
}
```

# Padrão de projeto DAO

- Data Access Object
- **Encapsula** as operações JDBC, **provendo acesso** de nível **mais alto** aos demais componentes da aplicação
- Pode ser usado em conjunto com outros padrões, como o Singleton

# Exercícios

---

# Trabalho de pesquisa

- Usando fontes de pesquisa atualizadas, discorra sobre os componentes básicos estudados em sala da API JDBC, ilustrando com exemplos a forma de utilização de cada um
- Complemente sua pesquisa, falando sobre a camada de persistência e os padrões de projeto DAO e Singleton, tipicamente usados para sua construção

# Trabalho de engenharia

- Construa uma camada de persistência que permita realizar o mapeamento objeto-relacional para um conjunto de classes de negócio a sua escolha
- Crie também classes de teste que lhe permitam assegurar que sua camada de persistência está plenamente funcional

# Roteiro para “instalação” do banco de dados

---

# Roteiro para instalação do BD

- Baixe <http://www.vqv.com.br/bd/mysql.zip> e descompacte em local de fácil acesso
- Na pasta **bin** e execute o arquivo de lote **start** (a janela que será aberta **não** pode ser fechada): se deu certo, o banco está no ar!
- Ative um **utilitário** de manipulação de banco de dados, como MySQL Query Browser, e teste sua instalação. Use **localhost**, usuário **root**, senha em branco e schema **test**

# Roteiro para instalação do BD

- **Ferramentas** para manipulação e administração do MySql estão em <http://dev.mysql.com/get/Downloads/MySQL>
- Para **parar** o banco de dados, execute o arquivo de lote **stop**, também na pasta bin
- **Evite** fechar a janela do banco diretamente, com o banco no ar!

# Disponibilização do driver

- Baixe <http://www.vqv.com.br/bd/driver.zip> e **descompacte** em local de fácil acesso
- No **Netbeans**, dentro de seu **projeto**, em **Bibliotecas**, adicione o *jar* que está na pasta descompactada no tópico anterior
- **Pronto!** Você está em condições de usar o driver JDBC para acessar, a partir do Java, o banco de dados MySql (para outros bancos, basta baixar o driver específico do banco)