APLICAÇÃO DA LINGUAGEM DE PROGRAMAÇÃO ORIENTADA À OBJETOS – ALPOO

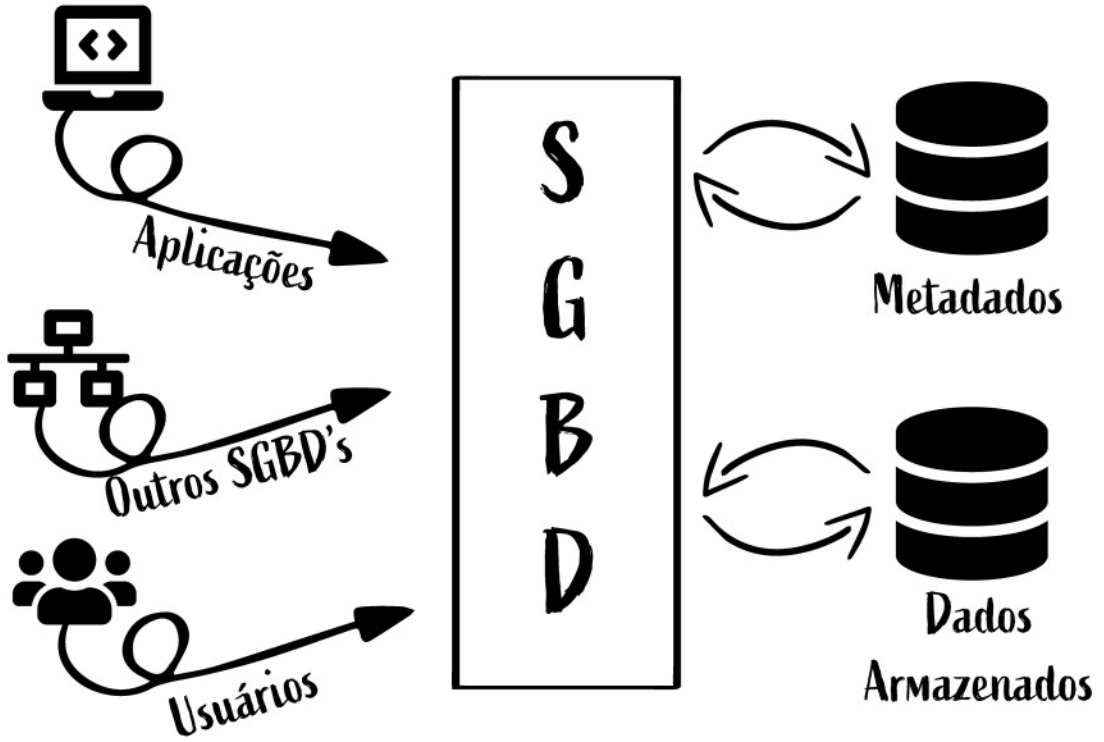
Aula 22 – Java Database Connectivity (JDBC)

Prof. Danilo Pereira - danilo.pereira1@docente.unip.br

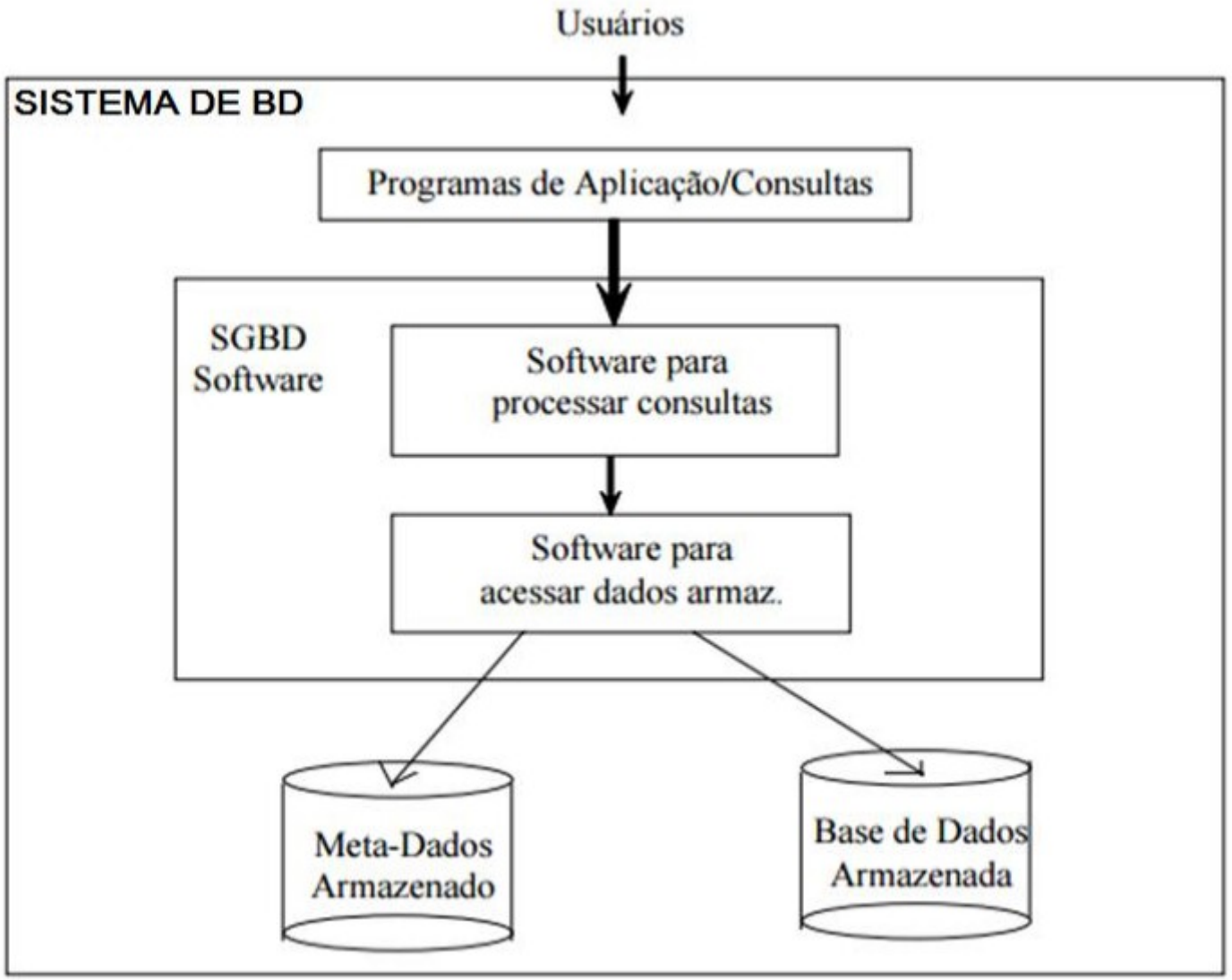
Sistema de Gerenciamento de Banco de Dados - SGBD

- Os SGBD são os elementos mais comuns para persistência de dados utilizados em aplicações comerciais, pois propiciam formas padronizadas para inserção, alteração, remoção e busca de dados.

- Portanto, é necessário verificar como as interfaces gráficas, quando acionadas pelo usuário, fazem o uso dos SGDBs para



Sistema de Gerenciamento de Banco de Dados - SGBD



Sistema de Gerenciamento de Banco de Dados - SGBD

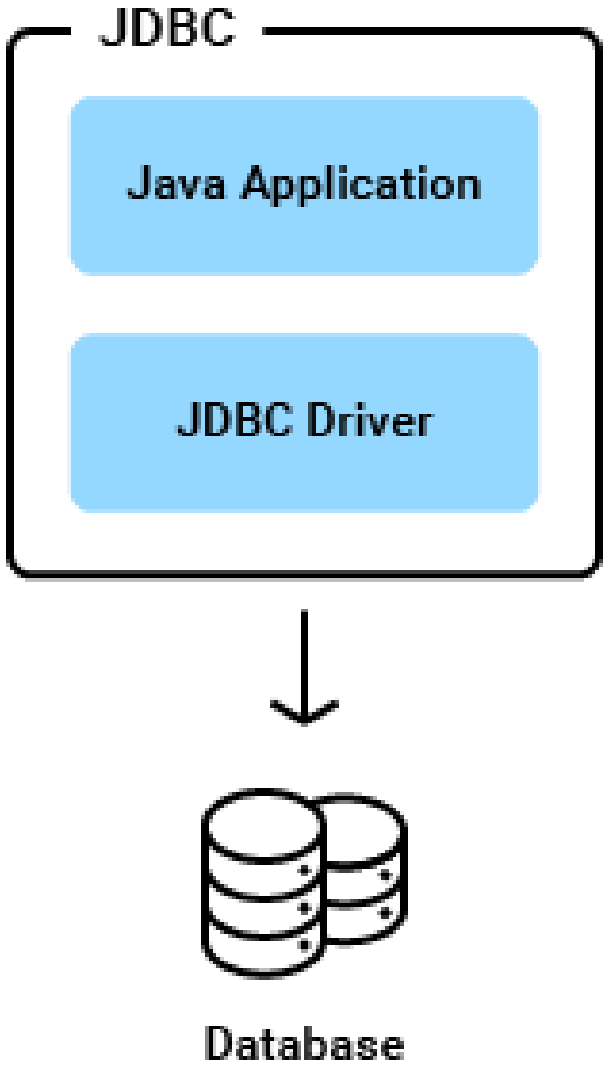
- Para utilizar os SGDBs em Java, especialmente em interfaces gráficas em Java Swing, é podemos utilizar o **Java Database Connectivity (JDBC)**.

- O JDBC consiste em um conjunto de classes que são incorporadas ao Java Development Kit (JDK)

Java Database Connectivity (JDBC)

- Para possibilitar o acesso a diversos SGDBs de forma padronizada sem a necessidade de se utilizar formas específicas para cada sistema de banco de dados.

- O JDBC é compatível com diversos sistemas de banco de dados, tais como:
  - MySQL
  - Oracle
  - PostgreSQL
  - SQLite



Principais falhas

- Banco de Dados não iniciado;
- Driver não Encontrado
- Driver incompatível;
- URL – Configurada de forma incorreta (nome banco, usuário, senha, servidor)
- Classe de conexão (inexistente, não instanciada, com erros)
- Usuário sem direito de acesso –
- Falta de tratamento de Exceções try .... catch

Arquitetura JDBC

- O JDBC utiliza as classes que controlam o driver que será utilizado para se conectar no banco de dados indicado. O primeiro passo, **que consiste em estabelecer a conexão, é necessário garantir que o JDBC conheça o SGBD**

String de conexão do Banco de Dados (URL)

- Para se conectar a um banco de dados, esteja ele implementado em qualquer SGBD, é necessário criar uma string de conexão, ou URL JDBC (Uniform Resource Locator JDBC).

- Essa string informará o “caminho” do banco e apresenta a seguinte sintaxe:

jdbc:<driver>:<detalhes da conexão>

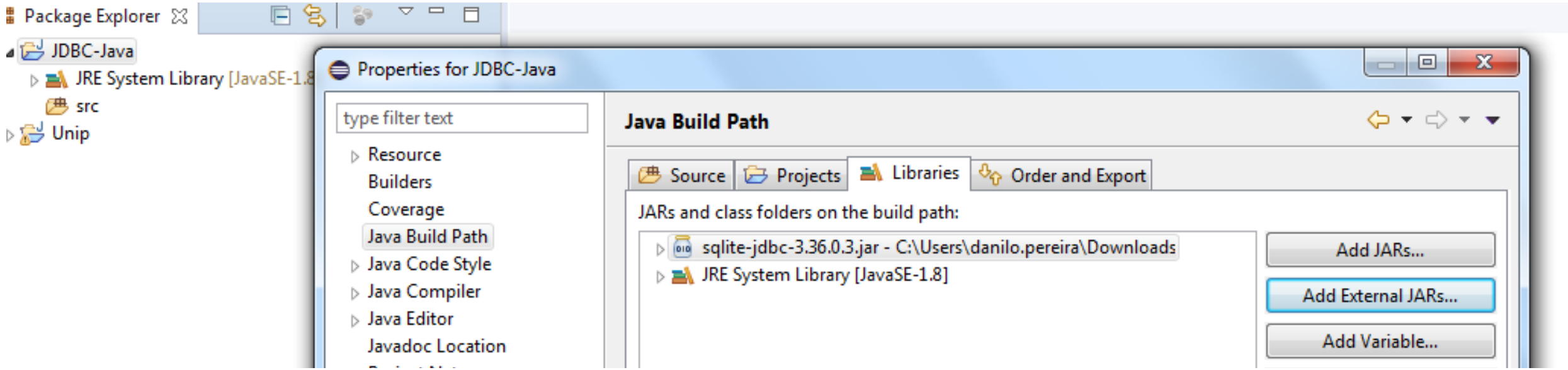
String de conexão do Banco de Dados (URL) - Exemplos

Banco de dados	URL JDBC
MySQL	jdbc:mysql://localhost:3306/nomeBancoDeDados
SQL Server	jdbc:sqlserver://localhost;databaseName=nomeBancoDeDados
Oracle	jdbc:oracle:thin@myserver:1521:nomeBancoDeDados

Usando SQLite em Java

1) Baixar o jar do SQLite (Verificar se existe versão maior) - https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc/3.36.0.3

2) Adicionar o jar no projeto



Verificando os drivers disponível para o seu

Antes de iniciar a criação da classe para conexão com o banco de dados, é extremamente importante garantir que o seu projeto faz o carregamento correto dos drives JDBC

Para isso, vamos criar uma classe Java, que exibirá uma lista de todos os drivers dos diferentes fabricantes (Oracle, MySQL, IBM, Postgresql, etc)

Verificando os drives disponíveis para o seu projeto Java

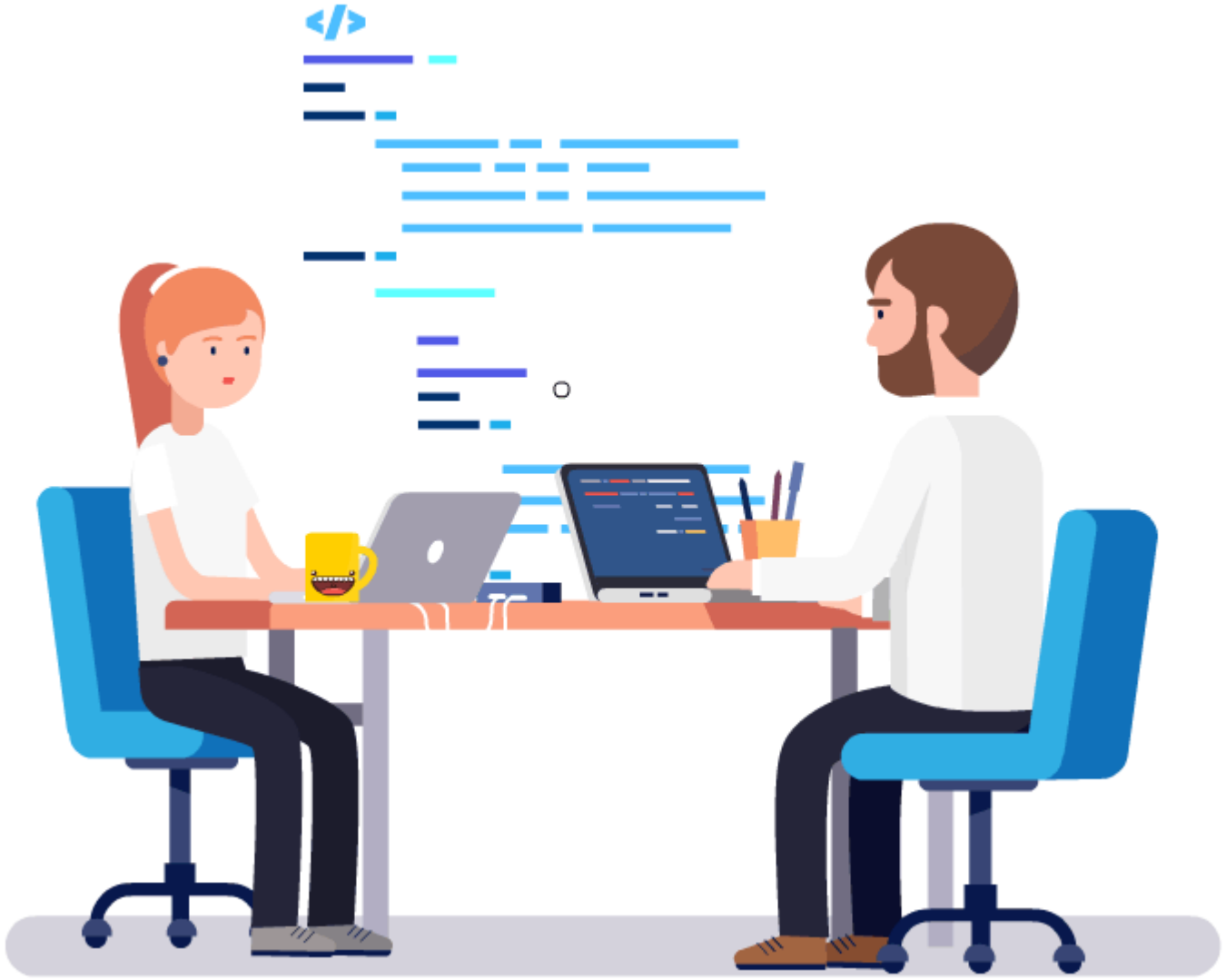
```
import java.sql.Driver;
import java.sql.DriverManager;
import java.util.Enumeration;

public class PrintJDBCDrivers {

    public static void main(String[] args) {
         System.out.println("List of loaded JDBC drivers");
        for (Enumeration<Driver> e = DriverManager.getDrivers();
        e.hasMoreElements();) {
            Driver driver = e.nextElement();
            print(driver);
        }
    }

    public static void print(Driver driver) {
        String className = driver.getClass().getName();
        int majorVersion = driver.getMajorVersion();
        int minorVersion = driver.getMinorVersion();
        System.out.println("---------------------------------------");
        System.out.println("Name Driver: " + className);
        System.out.println("Driver Major Version: " + majorVersion);
        System.out.println("Driver Minor Version: " + minorVersion);
        System.out.println("---------------------------------------");
    }
}
```

VAMOS PRATICAR ?



Conectando no banco de dados - Exemplo

```
public class Connect {

     public static void connect() {
               Connection conn = null;
          try {
               String url = "jdbc:sqlite:database_test.db";

               conn = DriverManager.getConnection(url);

               System.out.println("Connection to SQLite has been established.");

          } catch (SQLException e) {
               System.out.println(e.getMessage());
          } finally {
               try {
                    if (conn != null) {
                    conn.close();
                    }
               } catch (SQLException ex) {
               System.out.println(ex.getMessage());
               }
          }
     }

     public static void main(String[] args) {
         connect();
     }
}
```

Criando TABELAS

```
public class CreateTable {

    public static void createNewTable() {
        String url = "jdbc:sqlite:database_test.db";

        StringBuffer sql = new StringBuffer();
        sql.append("CREATE TABLE IF NOT EXISTS cliente (");
        sql.append("id integer PRIMARY KEY , ");
        sql.append("nome text NOT NULL, ");
        sql.append("idade integer, ");
        sql.append("cpf text NOT NULL, ");
        sql.append("rg text ");
        sql.append(")");

        try (Connection conn = DriverManager.getConnection(url);
          Statement stmt = conn.createStatement()) {
          stmt.execute(sql.toString());
        } catch (SQLException e) {
        System.out.println(e.getMessage());
        }
    }

    public static void main(String[] args) {
          createNewTable();
    }
}
```

Inserindo dados - Exemplo

```
public class InsertClient {

private Connection connect() {
        // SQLite connection string
        String url = "jdbc:sqlite:database_test.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }

    public void insert(Cliente cliente) {
        String sql = "INSERT INTO cliente(nome,idade, cpf, rg) VALUES(?,?,?,?)";

        try (Connection conn = this.connect();
                PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, cliente.getNome());
            pstmt.setInt(2, cliente.getIdade());
            pstmt.setString(3, cliente.getCpf());
            pstmt.setString(4, cliente.getRg());
            pstmt.executeUpdate();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
public static void main(String[] args) {
        InsertClient newClient = new InsertClient();

        Cliente c1 = new Cliente();
        c1.setNome("Danilo R. Pereira");
        c1.setIdade(37);
        c1.setCpf("111.111.111-11");
        c1.setRg("222.222.222-22");

        newClient.insert(c1);

}
```

Selecionando dados - Exemplo

```
public class SelectClient {

    private Connection connect() {
        String url = "jdbc:sqlite:database_test.db";
        Connection conn = null;
        try {
        conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
        System.out.println(e.getMessage());
        }
        return conn;
    }

    public void selectAll() {
        String sql = "SELECT id, nome, idade, cpf, rg FROM cliente";
        try (Connection conn = this.connect();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                System.out.println(rs.getInt("id") + "\t" + rs.getString("nome") + "\t" + rs.getInt("idade") + "\t" +
                rs.getString("cpf") + "\t" + rs.getString("rg"));
                }
            } catch (SQLException e) {
            System.out.println(e.getMessage());
            }
    }

    public static void main(String[] args) {
        SelectClient app = new SelectClient();
        app.selectAll();
    }
}
```

Atualizando dados - Exemplo

```
public class UpdateCliente {

    private Connection connect() {
        String url = "jdbc:sqlite:database_test.db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }

    public void update(int id, String name, Integer idade) {
        String sql = "UPDATE cliente SET nome = ? , + "idade = ? " + "WHERE id = ?";
        try (Connection conn = this.connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, name);
            pstmt.setInt(2, idade);
            pstmt.setInt(3, id);
          pstmt.executeUpdate();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }

    public static void main(String[] args) {
        UpdateCliente app = new UpdateCliente();
        app.update(1, "Danilo Rodrigues Pereira", 38);
    }
}
```

EXERCÍCIOS

1) Desenvolver uma classe em Java e com métodos CRUD (salvar, alterar, consultar e excluir):

Obs: Utilizar o banco de dados SQLite

Nome da tabela: PRODUTO
Colunas:
codigo INT PRIMARY KEY
nome VARCHAR
preco REAL
quantidade INT

Database Setup - MySQL

Certifique-se de ter instalado o servidor MySQL em sua máquina.

Como Baixar e Instalar MySQL 8.0 e MySQL Workbench no Windows 10

- https://www.youtube.com/watch?v=fmerTu7dWk8

Vamos primeiro criar um banco de dados com a seguinte instrução SQL:

create database swing_demo;

Conectando sua aplicação Java com o Banco de Dados

- Para que uma aplicação Java tenha acesso a um banco de dados relacional os seguintes passos devem ser atendidos:

  1) Habilitar o driver JDBC a partir da aplicação cliente;

  2) Estabelecer uma conexão entre a aplicação cliente e servidor do banco de dados;

  3) Montar e executar a instrução SQL desejada (create, insert, delete, select, etc...)

  4) Processar o resultado da consulta

Criando um projeto Java – JDBC + MySQL

Vamos criar um projeto Java simples no Eclipse IDE, nomeie esse projeto como **swing-registration-from-example**

Você pode usar este tutorial para criar um projeto Java simples no Eclipse IDE - https://www.wikihow.com/Create-a-New-Java-Project-in-Eclipse

Para conectar nosso programa Java ao banco de dados MySQL, precisamos incluir o driver JDBC MySQL, que é um arquivo JAR: mysql-connector-java-8.0.13-bin.jar.

Vamos baixar este arquivo jar e adicioná-lo ao classpath do seu projeto - https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.13

Verificando os drivers disponível para o seu

Antes de iniciar a criação da classe para conexão com o banco de dados, é extremamente importante garantir que o seu projeto faz o carregamento correto dos drives JDBC

Para isso, vamos criar uma classe Java, que exibirá uma lista de todos os drivers dos diferentes fabricantes (Oracle, MySQL, IBM, Postgresql, etc)

Verificando os drives disponível para o seu projeto

```java
public class PrintJDBCDrivers {
    public static void main(String[] args) {
        System.out.println("List of loaded JDBC drivers:");
        DriverManager.drivers().forEach(PrintJDBCDrivers::print);
    }

    public static void print(Driver driver) {
        String className = driver.getClass().getName();
        String moduleName = driver.getClass().getModule().getName();
        int majorVersion = driver.getMajorVersion();
        int minorVersion = driver.getMinorVersion();
        boolean jdbcCompliant = driver.jdbcCompliant();
        System.out.println("Driver Class Name: " + className);
        System.out.println("Driver Module Name: " + moduleName);
        System.out.println("Driver Major Version: " + majorVersion);
        System.out.println("Driver Minor Version: " + minorVersion);
        System.out.println("Driver jdbcCompliant: " + jdbcCompliant);
    }
}
```

Conectando com o banco de dados – Etapas

1) Após a criação do projeto, vamos criar uma classe chamada JDBCUtils e adicionar as importações: Connection, DriveManager, ResultSet, SQLException e Statements

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCUtils {



}
```

Conectando com o banco de dados – Etapas

2) A classe JDBCUtils deverá contar os seguintes métodos:

- **getConnection:** Responsável em criar a conexão Java -> DB
- **closeConnection:** Responsável em fechar a conexão Java -> DB (Importante !!!!)
- **closeStatement:** Responsável em fechar o executor de consulta e comandos SQL
- **closeResultSet:** Responsável em fechar o ResultSet (objeto que armazena os resultados da consultas e comandos SQL)
- **commit:** Responsável em fazer o commit das instruções sql enviadas para o DB
- **roolback:** Responsável em desfazer as instruções sql enviadas ao DB

Método getConnection

```
public static Connection getConnection() throws SQLException {
    String dbURL = "jdbc:mysql://localhost:3306/jdbc_demo_unip?"
            + "useTimezone=true&serverTimezone=UTC&useSSL=false";
    String userId = "root";
    String password = "      ";

    Connection conn = DriverManager.getConnection(dbURL, userId, password);

    conn.setAutoCommit(false);
    return conn;
}
```

Método closeConnection

```
    public static void closeConnection(Connection conn) {
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Método closeStatement

```
public static void closeStatement(Statement stmt) {
    try {
        if (stmt != null) {
            stmt.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Método closeResultSet

```
public static void closeResultSet(ResultSet rs) {
    try {
        if (rs != null) {
            rs.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Método commit

```
public static void commit(Connection conn) {
    try {
        if (conn != null) {
            conn.commit();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Método roolback

```
public static void rollback(Connection conn) {
    try {
        if (conn != null) {
            conn.rollback();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Método main – Testando a conexão

```
public static void main(String[] args) {
    Connection conn = null;
    try {
        conn = JDBCUtil.getConnection();
        System.out.println("Conexão com o banco de dados realizada com sucesso .");
    } catch (SQLException e) {
        System.err.println(" Erro ao conectar no banco de dados !!!");
        e.printStackTrace();
    } finally {
        JDBCUtil.closeConnection(conn);
    }
}
```

Criando tabela no MySQL

O objetivo principal de usar a API JDBC é manipular dados contidos em tabelas em um banco de dados.

Você pode manipular dados em tabelas usando as instruções SQL SELECT, INSERT, UPDATE e DELETE.

Para essa aula, vamos criar uma tabela chamada **person** com as seguintes características.

Column Name	Data Type	Length	Null Value Allowed	Comments
person_id	integer		No	Primary Key
first_name	string	20	No	
last_name	string	20	No	
gender	string	1	No	
dob	date		Yes	
income	double		Yes	

MySQL Database

```
create table person (
    person_id integer not null primary key,
    first_name varchar(20) not null,
    last_name varchar(20) not null,
    gender char(1) not null,
    dob datetime null,
    income double null
);
```

JDBC-Types-to-Java-Types Mapping

A API JDBC permite que você acesse e manipule dados em um ambiente Java, enquanto os dados são eventualmente armazenado em um banco de dados.

O banco de dados usa seus próprios tipos de dados, enquanto o Java usa seus próprios tipos.

Por isso, é necessário conhecer os mapeamentos entre os tipos de dados JDBC e os tipos de dados Java

JDBC-Types-to-Java-Types Mapping

Java Type	JDBC Type
String	CHAR, VARCHAR, or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT or BOOLEAN
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	BINARY, VARBINARY, or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
Clob	CLOB
Blob	BLOB
Array	ARRAY
Struct	STRUCT
Ref	REF
java.net.URL	DATALINK
Java class	JAVA_OBJECT

Executing SQL Statements

Você pode executar diferentes tipos de instruções SQL usando um driver JDBC e ela pode ser categorizada da seguinte maneira:

- **Uma declaração de linguagem de definição de dados (DDL):** exemplos de declarações DDL são CREATE TABLE, ALTER TABLE, etc.

- **Uma declaração de linguagem de manipulação de dados (DML):** exemplos de declarações DML são SELECT, INSERT, UPDATE, DELETE, etc.

- **Uma declaração de linguagem de controle de dados (DCL):** Exemplos de declarações DCL são GRANT e REVOKE.

- **Uma declaração de Linguagem de Controle de Transação (TCL): Exemplos** de declarações TCL são COMMIT, ROLLBACK, SAVEPOINT, etc.

Executing SQL Statements

Podemos executar instruções DDL, DML e DCL usando diferentes tipos de objetos de instrução JDBC.

Uma instância da **interface Statement** representa uma instrução SQL em um programa Java. Você pode executar TCL instruções usando os métodos de um objeto Connection.

Java usa três interfaces diferentes para representar SQL declarações em diferentes formatos:

- **Statement**
- **PreparedStatement**
- **CallableStatement**

LINK UTEIS

Conexão com Banco de Dados - https://www.ic.unicamp.br/~ra100621/class/2017.2/ALPOO_files/listaTiagoPaulo/ALPOO-Modulo%206-Exerc%C3%ADcios-MySQL.pdf

Banco de dados com Java: Introdução a JDBC - https://www.youtube.com/watch?v=db0vRGnfVuM

Aprenda Como Conectar JAVA com BANCO DE DADOS usando JDBC - https://www.youtube.com/watch?v=1v4iiTRFymE

EXERCÍCIOS

Criar uma tabela cliente no MySQL com as colunas: id, nome e e-mail.

Criar um JFrame para inserir registros nesta tabela.

Adicionar um botão para selecionar um cliente pelo nome e preencher os campos do formulário.

Adicionar um botão para alterar os dados do cliente que foi selecionado.

Adicionar um botão para excluir os dados do cliente que foi selecionado