### Conexión a una base de datos

Una **API (Application Programming Interface)** es un conjunto de reglas y protocolos que permite que diferentes aplicaciones de software se comuniquen entre sí. Esencialmente, una API define cómo las aplicaciones pueden interactuar, compartir datos y realizar acciones entre ellas, facilitando la integración y automatización de procesos entre sistemas.

### Funciones clave de una API:

1. **Interfaz**: Proporciona una serie de funciones que otros programas pueden utilizar para interactuar con un sistema o servicio.
2. **Abstracción**: Permite a los desarrolladores usar funciones de un sistema sin tener que conocer su implementación interna.
3. **Interoperabilidad**: Facilita la comunicación entre diferentes aplicaciones y plataformas, permitiendo la transferencia de datos y funciones.

### Ejemplos:

* Una API de un servicio de mapas, como Google Maps API, puede permitir que una aplicación de transporte acceda a mapas y funcionalidades de rutas sin tener que desarrollar su propio sistema de mapas.
* **JDBC (Java Database Connectivity)**: es la API estándar de Java para acceder a bases de datos relacionales. Funciona con cualquier base de datos que tenga un driver JDBC, como MySQL, PostgreSQL o Oracle.
* **Características principales de JDBC**:
  + Conexión a bases de datos relacionales.
  + Ejecución de sentencias SQL.
  + Recuperación y manipulación de datos.
  + Gestión de transacciones.

**Cónexión a una Base de Datos**

Nota: Antes de poder conectarse a una base de datos, es necesario cargar el driver JDBC que permitirá la comunicación entre Java y la base de datos específica. **Con JDBC 4.0 y posteriores, la carga del driver se hace automáticamente si está en el classpath, pero en versiones anteriores, se hacía manualmente con** Class.forName()**.**

*Código para la conexión a una Base de Datos:*

Connection con = DriverManager.getConnection(url, usuario, contraseña);

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM tabla");

while (rs.next()) {

System.out.println(rs.getString(1)); // Procesa el resultado

}

*Explicación del código:*

**Connection con = DriverManager.getConnection(url, usuario, contraseña);**

Establece una conexión a la base de datos utilizando la clase DriverManager con los parámetros url, usuario, y contraseña. La url especifica la dirección de la base de datos, y el usuario y la contraseña son las credenciales para acceder a ella.

El objeto Connection con representa la conexión activa.

Statement stmt = con.createStatement();

Crea un objeto Statement que se utiliza para ejecutar consultas SQL en la base de datos.

stmt es el objeto que lanzará las consultas SQL y gestionará los resultados.

ResultSet rs = stmt.executeQuery("SELECT \* FROM tabla");

Ejecuta la consulta SQL "SELECT \* FROM tabla" en la base de datos y obtiene un ResultSet que contiene los resultados.

"SELECT \* FROM tabla" es una consulta que selecciona todas las filas y columnas de la tabla especificada (tabla).

while (rs.next()) { System.out.println(rs.getString(1)); }

Este bucle while recorre cada fila del ResultSet.

rs.next() mueve el cursor al siguiente registro y devuelve true si hay más filas.

rs.getString(1) obtiene el valor de la primera columna de la fila actual como una cadena de texto y lo imprime. Esto permite ver el contenido de cada fila en la primera columna de la tabla.

Un ejemplo de una **URL de conexión** que puedes utilizar si tienes instalado un servidor de **MySQL** localmente en tu máquina:

String url = "jdbc:mysql://localhost:3306/mi\_base\_de\_datos";

String usuario = "root";

String contraseña = "tu\_contraseña";

### Explicación de la URL:

* jdbc:mysql://: Especifica el protocolo JDBC y el tipo de base de datos (MySQL).
* localhost:3306: localhost indica que el servidor está en la máquina local, y 3306 es el puerto predeterminado de MySQL.
* mi\_base\_de\_datos: El nombre de la base de datos a la que deseas conectarte. Debe ser una base de datos existente en tu servidor MySQL.

Si tienes MySQL en un servidor remoto, cambia localhost por la **dirección IP o el nombre de dominio** del servidor. Además, asegúrate de que el usuario y la contraseña sean válidos para acceder a esa base de datos.

### Conceptos adicionales

1. **Pooled Connections (Conexiones en Pool)**:
   * En aplicaciones empresariales de gran escala, es costoso crear y destruir conexiones de base de datos con frecuencia. Para optimizar esto, se usa un **pool de conexiones** que permite reutilizar conexiones abiertas.
2. **Drivers JDBC**:
   * Existen cuatro tipos de drivers JDBC:
     + **Driver Tipo 1 (Bridge Driver)**: Un puente entre JDBC y ODBC. No es muy eficiente y rara vez se usa.
     + **Driver Tipo 2 (Native API Driver)**: Usa APIs nativas específicas del sistema. Más eficiente pero depende del sistema operativo.
     + **Driver Tipo 3 (Network Protocol Driver)**: Se conecta a través de un middleware que traduce las llamadas JDBC.
     + **Driver Tipo 4 (Pure Java Driver)**: Es el más común y eficiente, ya que traduce las llamadas JDBC directamente a las llamadas de protocolo de la base de datos. No requiere instalación de software adicional.

### Ejemplo completo de conexión y consulta con JDBC

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class ConectarBaseDeDatos {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/miBaseDeDatos";

String usuario = "root";

String contraseña = "password";

try (Connection con = DriverManager.getConnection(url, usuario, contraseña);

Statement stmt = con.createStatement()) {

String consulta = "SELECT \* FROM productos";

ResultSet rs = stmt.executeQuery(consulta);

while (rs.next()) {

System.out.println("Producto: " + rs.getString("nombre"));

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}