# JDBC

API Java para acceso a bases de datos cuyas clases se encuentran localizadas en el paquete *java.sql*. La principal característica de JDBC es que permite independizar nuestro código Java de las peculiaridades de cada DBMS (Data Base Management System), de modo que nuestro programa Java pueda funcionar exactamente igual accediendo a una base de datos MySQL, una base de datos SQL Server, Oracle, etc.

JDBC se sitúa como una capa dentro de nuestra aplicación Java, capaz de comunicarse con el DBMS.

**En función del DBMS que soporte la base de datos, JDBC necesitará un driver o conector especial para Java que habrá que añadir al proyecto.**

Las clases principales del API JDBC son:

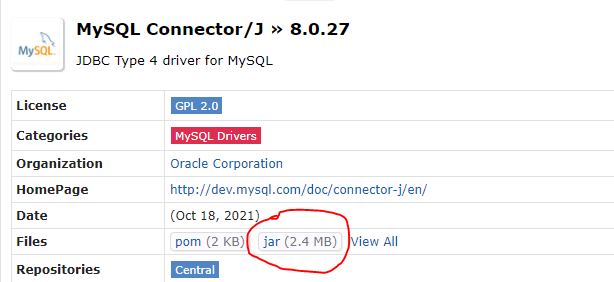
* **DriverManager:** proveedor de conexiones. Suministra un objeto Connection.
* **Connection:** representa una conexión con una base de datos.
* **Statement:** se comunica con el DBMS para enviar sentencias SQL. Es suministrado por el objeto Connection.
* **PreparedStatement:** sentencia SQL que ha sido precompilada. Aumenta el rendimiento cuando se va a utilizar varias veces la misma sentencia.
* **Resultset:** Conjunto de registros que resultan de la ejecución de una sentencia de tipo SELECT.

# El driver

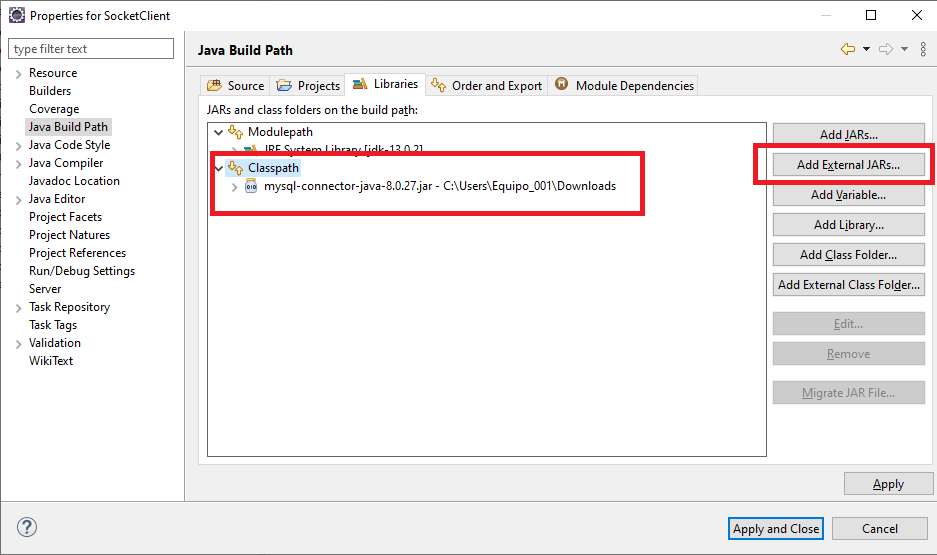
Antes de realizar el acceso desde Java a cualquier base de datos relacional a través de JDBC, es necesario realizar estas tres tareas:

* Descargar de Internet el driver específico para java suministrado por el DBMS.
* Importar el driver o conector en el proyecto Java que deba acceder a la base de datos.
* Cargar en memoria el driver, ejecutando el método Class.forName(...) desde el código Java.

Desde el repositorio central de Maven descarga el conector de MySQL para Java.



En cada proyecto Java que use JDBC deberás importar el archivo jar del driver que habrás obtenido al descomprimir el archivo descargado anteriormente. La importación la realizarás desde las propiedades del proyecto, Java BuildPath, AddExternalJARs.



La primera tarea de programación relacionada con el acceso a una base de datos será la carga en memoria del driver de la siguiente manera:

*public class Principal {*

*public static void main(String[] args) {*

*try {*

*Class.forName("com.mysql.cj.jdbc.Driver");*

*} catch (ClassNotFoundException e) {*

*System.out.println("No se ha encontrado el driver para MySQL");*

*return;*

*}*

*System.out.println("Se ha cargado el Driver de MySQL");*

*}*

*}*

Para lograr conectar a la base de datos FERRETERIA utilizaremos las siguientes clases:

* **DriverManager:** contiene información de todos los drivers o conectores cargados en memoria y actúa como proveedor, suministrando conexiones siempre y cuando tenga acceso a los drivers necesarios.
* **Connection:** representa una conexión con una base de datos determinada y es suministrada por el DriverManager.

Para establecer conexión con una base de datos se utiliza el método estático ***getConnection()*** de la clase ***DriverManager***, que retorna un objeto ***Connection***. El método *getConnection()* tiene el siguiente formato:

***Connection nombreObj = DriverManager.getConnection(String cadenaConexion, String usuario, String contraseña);***

Una **cadena de conexión** (***Connection String***) es un *String*que contiene información de acceso a una base de datos. El formato utilizado es diferente en función del proveedor de bases de datos. En términos generales, toda cadena de conexión se divide en las siguientes áreas:

* Proveedor, un valor que hace referencia al gestor de base de datos utilizado (jdbc:sqlserver, jdbc:derby, jdbc:mysql, etc.).
* Dirección IP o nombre del servidor de la BD.
* Nombre de la instancia. En SQL Server o MySQL un servidor puede contener varias instancias, cada instancia con sus propias bases de datos.
* Número de puerto. Necesario para establecer la conexión con el servidor. En MySQL es habitual que sea el 3306.
* Nombre de la base de datos o catálogo.

# Apertura y cierre de la conexión

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class Principal {

public static void main(String[] args) {

// Paso 1: Cargar el driver

try {

Class.forName("com.mysql.cj.jdbc.Driver");

} catch (ClassNotFoundException e) {

System.out.println("No se ha encontrado el driver para MySQL");

return;

}

System.out.println("Se ha cargado el Driver de MySQL");

// Paso 2: Establecer conexión con la base de datos

String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";

String user = "root";

String pass = "curso";

Connection con;

try {

con = DriverManager.getConnection(cadenaConexion, user, pass);

} catch (SQLException e) {

System.out.println("Error en la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha establecido la conexión con la Base de datos");

// Paso 3: Interactuar con la BD (todavía pendiente)

// Paso 4: Cerrar la conexión

try {

con.close();

} catch (SQLException e) {

System.out.println("No se ha podido cerrar la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha cerrado la base de datos");

}

}

# Interactuando con la base de datos (listado de clientes)

// Paso 3: Interactuar con la BD

try {

Statement sentencia = con.createStatement();

ResultSetrs = sentencia.executeQuery("SELECT \* FROM CLIENTE");

while (rs.next()) {

System.out.print(rs.getString("NIF"));

System.out.print(" - ");

System.out.print(rs.getString("NOMBRE"));

System.out.print(" - ");

System.out.print(rs.getString("TLF"));

System.out.println(); // Retorno de carro

}

} catch (SQLException e) {

System.out.println("Error al realizar el listado de productos");

System.out.println(e.getMessage());

}

# Moverse por el conjunto de registros: el cursor

El objeto ResultSet utiliza un cursor como indicador del próximo registro a leer y provee de varios métodos destinados a modificar la posición del cursor, por ejemplo ir al primer registro, ir al último registro, ir a una posición concreta, etc.

Veamos los métodos de la clase ResultSet que tienen que ver con la posición del cursor:

### boolean absolute(int fila)

Si le pasamos como argumento un valor positivo, por ejemplo 3, sitúa el puntero en el registro 3, es decir, tres posiciones por delante del primer registro. Si le pasamos un valor negativo, por ejemplo -3, se mueve tres posiciones desde el final del último registro.

Retorna true si existe el registro al que hemos intentado movernos, de lo contrario retorna false y se sitúa por delante del último registro. También puede situarse por delante del primer registro si hemos usado un valor negativo.

### void afterLast()

Sitúa el puntero por detrás del último registro (end of file).

### void beforeFirst()

Sitúa el puntero por delante del primer registro (botton of file).

### boolean first()

Sitúa el puntero en el primer registro. Retorna true si existe el primer registro, retornará false si no hay ningún registro.

### boolean last()

Coloca el puntero en el último registro. Retorna true si existe el último registro, retornará false si no hay ningún registro.

### boolean next()

Mueve el puntero al siguiente registro. Si ha alcanzado el final de fichero (end of file) retornará false, de lo contrario retornará true.

### boolean previous()

Mueve el puntero al anterior registro. Si el puntero se ha posicionado por delante del primer registro (botton of file) retornará false, de lo contrario retornará true.

### boolean relative(int filas)

Si pasamos como argumento un valor positivo, por ejemplo 3, mueve el puntero tres posiciones hacia delante. Si pasamos un valor negativo, por ejemplo -3, mueve el puntero tres posiciones hacia atrás.

Retorna true si existe el registro al que hemos intentado movernos, de lo contrario retorna false y se sitúa por delante del último registro o por delante del primer registro si hemos usado un valor negativo.

### boolean isBeforeFirst()

Informa si el puntero se encuentra por delante del primer registro, en cuyo caso retornará true.

### boolean isAfterLast()

Informa si el puntero se encuentra por detrás del último registro, en cuyo caso retornará true.

Para poner en práctica alguno de estos métodos puedes modificar la tercera parte del programa anterior (Paso 3: Interactuar con la BD) para recorrer los registros en orden inverso, es decir, situarse en el último registro y utilizar varias veces el método previous() para leer desde el último registro hasta el primero.

// Paso 3: Interactuar con la BD

try {

Statement sentencia = con.createStatement();

ResultSet rs = sentencia.executeQuery("SELECT \* FROM CLIENTE");

boolean hayRegistros = rs.last();

while (hayRegistros) {

System.out.print(rs.getString("NIF"));

System.out.print(" - ");

System.out.print(rs.getString("NOMBRE"));

System.out.print(" - ");

System.out.print(rs.getString("TLF"));

System.out.println(); // Retorno de carro

hayRegistros = rs.previous();

}

} catch (SQLException e) {

System.out.println("Error al realizar el listado de clientes");

System.out.println(e.getMessage());

}

# Trabajando con distintos tipos de cursores

El método createStatement() puede recibir dos argumentos, que condicionan las operaciones que podemos hacer con nuestro conjunto de registros.

*Statement createStatement(int resultSetType, int resultSetConcurrency);*

El argumento resultSetType define el tipo de cursor. Podemos asignar una de las siguientes constantes con nombre como valor para este argumento:

* ResultSet.TYPE\_FORWARD\_ONLY: permite un solo recorrido y solamente hacia adelante.
* ResultSet.TYPE\_SCROLL\_INSENSITIVE: permite recorrer el cursor varias veces y en cualquier sentido, pero no es sensible a los cambios realizados por otros usuarios.
* ResultSet.TYPE\_SCROLL\_SENSITIVE: permite recorrer el cursor varias veces y en cualquier sentido, además es sensible a los cambios realizados por otros usuarios. Si trabajamos en entorno multiusuario estamos viendo en tiempo real los cambios realizados por otros usuarios de la aplicación.

El argumento resultSetConcurrency define el tipo de concurrencia. Podemos asignar una de las siguientes constantes con nombre:

* ResultSet.CURSOR\_READ\_ONLY: cursor de solo lectura, no podemos añadir nuevos registros o modificar los existentes.
* ResultSet.CURSOR\_UPDATABLE: cursor de lectura y escritura. Podemos actualizar los registro existentes e incluso añadir nuevos.

# Modificar la dirección y el teléfono del cliente con NIF = "43434343A".

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class PrincipalActualizar {

public static void main(String[] args) {

// Paso 1: Cargar el driver

try {

Class.forName("com.mysql.jdbc.Driver");

} catch (ClassNotFoundException e) {

System.out.println("No se ha encontrado el driver para MySQL");

return;

}

System.out.println("Se ha cargado el Driver de MySQL");

// Paso 2: Establecer conexión con la base de datos

String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";

String user = "root";

String pass = "amelia";

Connection con;

try {

con = DriverManager.getConnection(cadenaConexion, user, pass);

} catch (SQLException e) {

System.out.println("No se ha podido establecer la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha establecido la conexión con la Base de datos");

// Paso 3: Interactuar con la BD

try {

Statement sentencia = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSetrs = sentencia.executeQuery("SELECT \* FROM CLIENTE WHERE NIF = '43434343A'");

booleanexiste = rs.next();

if (existe) {

rs.updateString("DOMICILIO", "C/ PEÑAGRANDE, 52");

rs.updateString("TLF", "123456789");

rs.updateRow();

}

} catch (SQLException e) {

System.out.println("Error al actualizar los datos del cliente");

System.out.println(e.getMessage());

}

// Paso 4: Cerrar la conexión

try {

con.close();

} catch (SQLException e) {

System.out.println("No se ha podido cerrar la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha cerrado la base de datos");

}

}

# Añadir un nuevo registro

importjava.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class PrincipalAlta {

public static void main(String[] args) {

// Paso 1: Cargar el driver

try {

Class.forName("com.mysql.jdbc.Driver");

} catch (ClassNotFoundException e) {

System.out.println("No se ha encontrado el driver para MySQL");

return;

}

System.out.println("Se ha cargado el Driver de MySQL");

// Paso 2: Establecer conexión con la base de datos

String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";

String user = "root";

String pass = "amelia";

Connection con;

try {

con = DriverManager.getConnection(cadenaConexion, user, pass);

} catch (SQLException e) {

System.out.println("No se ha podido establecer la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha establecido la conexión con la Base de datos");

// Paso 3: Interactuar con la BD

try {

Statement sentencia = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSetrs = sentencia.executeQuery("SELECT \* FROM CLIENTE");

**rs.moveToInsertRow();**

**rs.updateString("NIF", "12345678Z");**

**rs.updateString("NOMBRE", "HIPOLITO DURAN CARLOS");**

**rs.updateString("DOMICILIO", "C/ PEZ VOLADOR, 52");**

**rs.updateString("TLF", "912222222");**

**rs.updateString("CIUDAD", "MADRID");**

**rs.insertRow();**

} catch (SQLException e) {

System.out.println("Error al añadir el nuevo cliente");

System.out.println(e.getMessage());

}

// Paso 4: Cerrar la conexión

try {

con.close();

} catch (SQLException e) {

System.out.println("No se ha podido cerrar la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha cerrado la base de datos");

}

}

# Eliminar cliente con NIF = "12345678Z".

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class PrincipalBorrar

public static void main(String[] args) {

// Paso 1: Cargar el driver

try {

Class.forName("com.mysql.jdbc.Driver");

} catch (ClassNotFoundException e) {

System.out.println("No se ha encontrado el driver para MySQL");

return;

}

System.out.println("Se ha cargado el Driver de MySQL");

// Paso 2: Establecer conexión con la base de datos

String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";

String user = "root";

String pass = "amelia";

Connection con;

try {

con = DriverManager.getConnection(cadenaConexion, user, pass);

} catch (SQLException e) {

System.out.println("No se ha podido establecer la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha establecido la conexión con la Base de datos");

// Paso 3: Interactuar con la BD

try {

Statement sentencia = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE);

ResultSetrs = sentencia.executeQuery("SELECT \* FROM CLIENTE WHERE NIF = '12345678Z'");

booleanexiste = rs.next();

if (existe) {

rs.deleteRow();

}

} catch (SQLException e) {

System.out.println("Error al eliminar al cliente");

System.out.println(e.getMessage());

}

// Paso 4: Cerrar la conexión

try {

con.close();

} catch (SQLException e) {

System.out.println("No se ha podido cerrar la conexión con la BD");

System.out.println(e.getMessage());

return;

}

System.out.println("Se ha cerrado la base de datos");

}

}