UT03_01 Conexión a base de datos mediante Driver Manager y cadenas de conexión JDBC

1. JDBC

JDBC (Java DataBase Connectivity) es la API que permite la conexión de un programa Java y una base de datos relacional. Se encuentra dentro del paquete java.sql.

Incluye clases e interfaces que permiten el acceso a la bases de datos para ejecutar consultas, actualizaciones, ejecutar procedimientos, etc.

Algunas de las clases e interfaces de JDBC son:

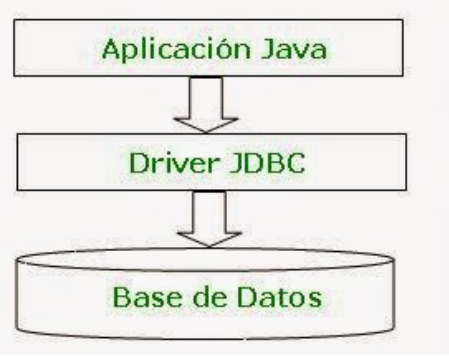
Clase / Interface	Función
Clase DriverManager	Establece la conexión con la base de datos
Interface Connection	Representa una conexión con la BD
Interface Statement	Ejecución de consultas SQL
Interface PreparedStatement	Ejecución de consultas preparadas y procedimientos almacenados
Interface ResultSet	Manipulación de registros en consultas de tipo Select
Interface ResultSetMetadata	Proporciona información sobre la estructura de los datos.

El objetivo de las interfaces de JDBC es definir como trabajar con la base de datos: como establecer la conexión, como ejecutar una consulta, etc.

Para poder ejecutar nuestro programa necesitamos las clases que implementen estas interfaces.

1.1 Driver JDBC

Llamamos **DRIVER** al conjunto de clases que implementan las interfaces JDBC. El driver proporciona la comunicación entre la aplicación Java y la base de datos



Cada tipo de bases de datos (Oracle, MySQL, PostGreSQL, etc) tienen su propio driver.

Los drivers los proporcionan los fabricantes de las bases de datos. Normalmente se descarga desde la página web del fabricante. Por lo tanto, el primer paso para trabajar con bases de datos desde Java es conseguir el driver adecuado.

En nuestro caso utilizaremos MySQL cuyo driver vamos a descargar desde:

http://dev.mysql.com/downloads/connector/j/

Una vez descargado lo descomprimimos y **el .jar que contiene lo debemos añadir como librería de clases a nuestro proyecto**.

1.2 ACCESO A BASES DE DATOS CON JDBC

Una vez incorporado el driver como librería de la aplicación, se deben seguir los siguientes pasos para acceder a una base de datos:

1. Importar los paquetes: Normalmente es suficiente con la sentencia *import java.sql.*;*

2. Cargar el driver: El driver se debe cargar para poder utilizarlo. Esto lo realiza el método estático *forName()* de la clase *Class*.

```
Class.forName(String driver);

Driver JDBC para MySQL:

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

Lanza una excepción **ClassNotFoundException**

3. Crear la conexión: Esto lo realiza el método estático *getConnection()* de la clase *DriveManager*.

```
DriverManager.getConnection(String url);
```

url es un String que nos permite localizar la base de datos.

Normalmente se compone de tres campos:

jdbc:tipoBD:datos_de_conexion

Para MySQL, el formato es:

"jdbc:mysql://*ordenador_donde_está_la_base_de_datos*/*base_de_datos*".

Deberemos indicar el nombre o IP del ordenador en el que se encuentra el servidor de base de datos y a continuación el nombre de la base de datos.

Una sobrecarga del método getConnection acepta, además de la url, el usuario y contraseña de la base de datos.

Este método devuelve un objeto que implementa la interfaz Connection.

Por ejemplo, si tenemos corriendo en nuestro ordenador el servidor de bases de datos, y nos queremos conectar a una base de datos llamada *prueba* que tiene un usuario *root* y contraseña *1daw* escribiremos:

```
Connection conexion =

DriverManager.getConnection("jdbc:mysql://localhost/prueba","root","1daw");
```

4. Ejecutar una consulta: Las consultas se manejan mediante un objeto que implementa la interface *Statement*. Antes de poder ejecutar una consulta debemos crear el objeto.

El objeto se crea utilizando el método *createStatement()* de *Connection*.

```
Statement s = conexion.createStatement();
```

Una vez creado podemos utilizar algunos de los métodos de Statement para enviar la consulta a la BD. Algunos de estos métodos son:

Método	Descripción
boolean execute(String sentenciaSQL);	Si es una consulta de acción (Insert, Delete, Update) devuelve false indicando que no se generan resultados. Si es una consulta de selección (Select) devuelve true.
int executeUpdate(String sentenciaSQL);	Envía una consulta de acción. Devuelve el número de registros afectados por dicha acción.
ResultSet executeQuery(String sentenciaSQL);	Envía una consulta de selección. Devuelve un objeto ResultSet con los datos obtenidos de la consulta.

Por ejemplo, si en la base de datos tenemos una tabla llamada *persona*, una consulta de todos los registros de la tabla sería:

```
ResultSet rs = s.executeQuery("select * from persona");
```

5. Manipular el resultado de la consulta: El objeto de tipo ResultSet devuelto por la consulta de selección dispone de un cursor para desplazarse por los registros y de métodos para acceder al valor de cada campo.

Desplazarse por los registros de un ResultSet

Una vez obtenido, su cursor se sitúa justo antes del primer registro obtenido.

Por defecto, **un ResultSet es un objeto de sólo avance y solo lectura**. El recorrido se hará siempre desde el principio hacia delante y los campos no podrán ser modificados. Se pueden crear también ResultSets desplazables de lectura/escritura.

El método next() permite desplazarnos por los registros. Una llamada a este método desplaza el cursor al siguiente registro. Devuelve false si se ha llegado al final y true en caso contrario.

```
while (rs.next()) {

.....

}
```

Acceder a los campos del resultado de una consulta

Los campos del registro apuntado por el cursor se pueden acceder mediante dos grupos de métodos. xxx es cualquier tipo de dato básico Java más String, Date y Object. Se deberá utilizar el método adecuado al tipo de dato que contiene el campo. Si no conocemos el tipo de dato podemos utilizar getObject().

Método	Descripción
xxx getXxx(int posición);	Obtiene el valor del campo que ocupa la posición indicada. La primera posición es la 1.
xxx getXxx(String nombreCampo);	Obtiene el valor del campo que coincide con el nombre indicado.

Por ejemplo, si la tabla *persona* contiene 3 campos en este orden: id de tipo int, nombre de tipo String y fecha de tipo Date:

```
while (rs.next()) {

        System.out.println(rs.getInt("Id") + " " + rs.getString(2) + " " + rs.getDate(3));

}
```

Además de estos métodos, la interfaz ResultSet proporciona una gran cantidad de métodos para acceder y manejar los resultados de la consulta entre ellos:

boolean isFirst() que devuelve true si el cursor apunta al primer registro.

boolean isLast() que devuelve true si el cursor apunta al último.

int getRow() que devuelve la posición del registro actual, siendo 1 la posición del primer registro, etc.

http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html

6. Desconexión de la base de datos: Permite liberar recursos de memoria y CPU. El cierre de una conexión se realiza mediante el método *close()* de la interfaz *Connection*.

```
conexion.close();
```

La interfaz Statement también dispone de un método close() para liberar los recursos utilizados por el objeto. Esto debe hacerse antes de cerrar la conexión.

Cuando se cierra una conexión todos los objetos Statement se cierran automáticamente.

La mayoría de métodos usados el la API JDBC lanzan una excepción de tipo **SQLException** que se debe capturar.

Ejemplos de acceso a bases de datos MySQL con Java y JDBC

Para realizar los siguientes ejemplos de acceso a una base de datos supondremos lo siguiente:

El servidor de base de datos debe estar arrancado y suponemos que utiliza el puerto por defecto (3306)

La base de datos que vamos a utilizar se llama *prueba* con un usuario *root* y contraseña *1daw*.
La base de datos tiene una tabla *persona* con tres campos:
id: smallint auto_increment primary key
nombre: varchar(60)
apellidos: varchar(60)
nacimiento: date

Ejemplo 1: Conexión a una base de datos MySQL y consulta de una tabla.

```
package es.cifpcm;


/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
import java.sql.*;
/**
 *
 * @author inmav
 */
public class EjemploAccesoBD1 {


/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
```

```
Connection conexion = null;
try {
// Cargar el driver
Class.forName("com.mysql.jdbc.Driver");

// Se obtiene una conexión con la base de datos.
// En este caso nos conectamos a la base de datos prueba
// con el usuario root y contraseña 1daw
conexion = DriverManager.getConnection("jdbc:mysql://localhost/prueba", "root",
"2daw");

// Se crea un Statement, para realizar la consulta
Statement s = conexion.createStatement();

// Se realiza la consulta. Los resultados se guardan en el ResultSet rs
ResultSet rs = s.executeQuery("select * from persona");

// Se recorre el ResultSet, mostrando por pantalla los resultados.
while (rs.next()) {
System.out.println(rs.getInt("Id") + " " + rs.getString(2) + " " + rs.getString(3)+"
" + rs.getDate(4));
}
} catch (SQLException | ClassNotFoundException e) {
System.out.println(e.getMessage());
} finally { // Se cierra la conexión con la base de datos.
try {
if (conexion != null) {
conexion.close();
}
```

```
} catch (SQLException ex) {

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

}

}

}

}
```

Ejemplo 2: Crear una tabla e insertar datos. **¡¡¡No muy bueno crear tablas en bases de datos directamente desde java....)!!!**

Crearemos la tabla contactos dentro de la base de datos prueba.

```
/*

 * To change this license header, choose License Headers in Project Properties.

 * To change this template file, choose Tools | Templates

 * and open the template in the editor.

 */

package es.cifpcm;

import java.sql.*;


/**

 *

 * @author inmav

 */

public class EjemploAccesoBD2 {

public static void main(String[] args) {

Connection conexion = null;

try {

// Cargar el driver

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


// Se obtiene una conexión con la base de datos.
```

```
conexion = DriverManager.getConnection("jdbc:mysql://localhost/prueba", "root", "2daw");


// Se crea un Statement, para realizar el query

Statement s = conexion.createStatement();


//se crea una tabla nueva

s.executeUpdate("CREATE TABLE contacto (id INT AUTO_INCREMENT, PRIMARY KEY(id), nombre VARCHAR(20), apellidos VARCHAR(20), telefono VARCHAR(20))");


//Los datos que vamos a insertar los tenemos en 3 arrays

String nombres[] = {"Juan", "Pedro", "Antonio"};

String apellidos[] = {"Gomez", "Lopez", "Alvarez"};

String telefonos[] = {"987452154", "989654125", "985321478"};


//se insertan datos en la tabla

for (int i = 0; i < nombres.length; i++) {

s.executeUpdate("INSERT INTO contacto (nombre, apellidos, telefono) VALUES ('"+ nombres[i] + "','" + apellidos[i] + "','" + telefonos[i] + "' )");

}


// Se realiza una consulta sobre la tabla contacto.

ResultSet rs = s.executeQuery("select * from contacto");


// Se recorre el ResultSet, mostrando por pantalla los resultados.

while (rs.next()) {

System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " +

rs.getString(3) + " " + rs.getString(4));

}

} catch (SQLException e) {
```

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

} catch (ClassNotFoundException e){

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

} finally { // Se cierra la conexión con la base de datos.

try {

if (conexion != null) {

conexion.close();

}

} catch (SQLException ex) {

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

}

}

}

}
```

En la instrucción:

```
s.executeUpdate("INSERT INTO contacto (nombre, apellidos, telefono)

                VALUES ('" + nombres[i] + "','" + apellidos[i] + "','" + telefonos[i] + "' )");
```

Las variables de tipo String que corresponden a los datos de tipo VARCHAR de la tabla deben ir entre comillas simples.

Si la instrucción se hiciera con datos fijos sería:

INSERT INTO contacto (nombre, apellidos, telefono) VALUES ('Juan', 'Gomez', '987452154');

Las comillas no se escriben para datos numéricos.

Ejemplo 3: Modificar el teléfono del primer contacto de la tabla persona con nombre *Juan*.

```
import java.sql.*;

public class EjemploAccesoBD3 {

public static void main(String[] args) {

        Connection conexion = null;

        int id;

        try {
```

```
            // Cargar el driver
            Class.forName("com.mysql.jdbc.Driver");

            // Se obtiene una conexión con la base de datos.
conexion = DriverManager.getConnection("jdbc:mysql://localhost/prueba", "root", "2daw");

            // Se crea un Statement, para realizar el query
            Statement s = conexion.createStatement();

            // Se realiza la consulta
            // Queremos obtener el id del primer contacto con nombre Juan
            ResultSet rs = s.executeQuery("SELECT id FROM contacto WHERE nombre='Juan'");


            if(rs.next()){ //Si rs.next() devuelve true significa que al menos hay un registro
                    id = rs.getInt("id"); //se obtienen su id
                    //se actualiza el registro
                    s.executeUpdate("UPDATE contacto SET telefono='987654321' WHERE id="+id);
             }
            } catch (SQLException e) {
             System.out.println(e.getMessage());
            } catch (ClassNotFoundException e) {
             System.out.println(e.getMessage());
            } finally { // Se cierra la conexión con la base de datos.
             try {
            if (conexion != null) {
             conexion.close();
            }
             } catch (SQLException ex) {
                    System.out.println(ex.getMessage());
             }
            }
```

```
}

}
```

ResultSet desplazables y modificables

Se pueden crear ResultSet con más prestaciones que las que tienen los que hemos creado hasta ahora. Recuerda que los creados por defecto solo se pueden recorrer desde el principio hasta el final y no permiten modificar los datos.

Para ello tenemos que modificar la forma de crear el Statement.

En lugar de crear un Statement así: Statement s = conexion.createStatement();

Utilizaremos esta versión del método createStatement():

```
Statement createStatement(int tipoResultSet, int concurrenciaResultSet);
```

tipoResulSet puede ser uno de estos valores:

ResultSet.TYPE_FORWARD_ONLY	ResultSet de solo avance. Es el valor por defecto
ResultSet.TYPE_SCROLL_INSENSITIVE	ResultSet desplazable en ambas direcciones. No refleja los cambios en los datos que se puedan producir en la base de datos.
ResultSet.TYPE_SCROLL_SENSITIVE	ResultSet desplazable en ambas direcciones. Refleja los cambios en los datos que se puedan producir en la base de datos.

concurrenciaResultSet puede ser uno de estos valores:

ResultSet.CONCUR_READ_ONLY	Campos de solo lectura. Es el valor por defecto
ResultSet.CONCUR_UPDATABLE	Campos modificables.

Además de los métodos utilizados por los ResultSet por defecto, algunos de los métodos que se pueden usar son los siguientes:

Método	Descripciόm
boolean first()	Desplaza el cursor al primer registro. Devuelve true si estamos ante una fila válida y false en caso contrario.
void beforeFirst()	Desplaza el cursor a la posición situada antes del primer registro.
boolean last()	Desplaza el cursor al último registro. Devuelve true si estamos ante una fila válida y false en caso contrario.
void afterLast()	Desplaza el cursor a la posición situada después del último registro.
boolean absolute(int pos)	Desplaza el cursor a la posición indicada por pos. Devuelve true si

	estamos ante una fila válida y false en caso contrario.

Para modificar un campo del ResultSet, utilizamos métodos updateXxx donde Xxx es el tipo de dato del campo que vamos a actualizar.

Por ejemplo: modificamos el campo 2 (de tipo String) del registro 2 de la tabla persona.

```
rs.absolute(2); //nos situamos en el registro a modificar

rs.updateString(2, "Ana Lozano");// Modificamos el campo 2. Nuevo valor: "Ana Lozano"

rs.updateRow(); // se actualiza el registro.
```

http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html

Ejemplo de ResultSet desplazable y modificable

En este ejemplo se crea un ResultSet que se va recorrer de forma inversa, desde el final al principio. Además se va a utilizar para modificar un registro de la tabla persona.

```
import java.sql.*;

public class EjemploAccesoBD6 {

public static void main(String[] args) {

        Connection conexion = null;

        try {

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

       conexion =

        DriverManager.getConnection("jdbc:mysql://localhost/prueba", "root", "2daw");


       //Indicamos que el ResultSet será desplazable y modificable

       Statement s = conexion.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,

       ResultSet.CONCUR_UPDATABLE);


       ResultSet rs = s.executeQuery("select * from persona");


       //Recorrer el ResultSet desde el final hasta el principio

       rs.afterLast();

       while (rs.previous()) {
```

```
 System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getDate(3));
}

// modificar el campo nombre (2) del registro 2 del ResultSet
// el cambio también se produce en la base de datos
rs.absolute(2);
rs.updateString(2, "Ana Lozano");
rs.updateRow();

//Recorrer el ResultSet para comprobar la modificación.
//Para recorrer el ResultSet desde el principio hasta el final nos debemos situar
//de nuevo al principio
rs.beforeFirst();
while (rs.next()) {
 System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getDate(3));
}
 } catch (SQLException e) {
System.out.println(e.getMessage());
 } catch (ClassNotFoundException e) {
System.out.println(e.getMessage());
 } finally {
try {
 if (conexion != null) {
conexion.close();
 }
} catch (SQLException ex) {
 System.out.println(ex.getMessage());
```

```
      }

       }

}

}
```

Para **insertar filas** el ResultSet tiene el método moveToInsertRow() que nos lleva a una fila vacía para que llenemos sus campos mediante métodos updateXxx y finalmente la insertemos en la tabla mediante el método insertRow().

El esquema básico para insertar una fila utilizando ResultSet:

```
rs.moveToInsertRow(); //posicionarse en la fila vacía

rs.updateXxx(campo, valor) // hacer los updates de los campos

rs.updateXxx(campo, valor);

…..

rs.insertRow();
//finalmente insertar la nueva fila en la tabla.
```