12.C. Conectando con la base de datos.

Sitio:	Formación Profesional a Distancia
Curso:	Programación
Libro:	12.C. Conectando con la base de datos.
Imprimido por:	Iván Jiménez Utiel
Día:	domingo, 17 de mayo de 2020, 16:27

Tabla de contenidos

1. Establecimiento de conexiones.

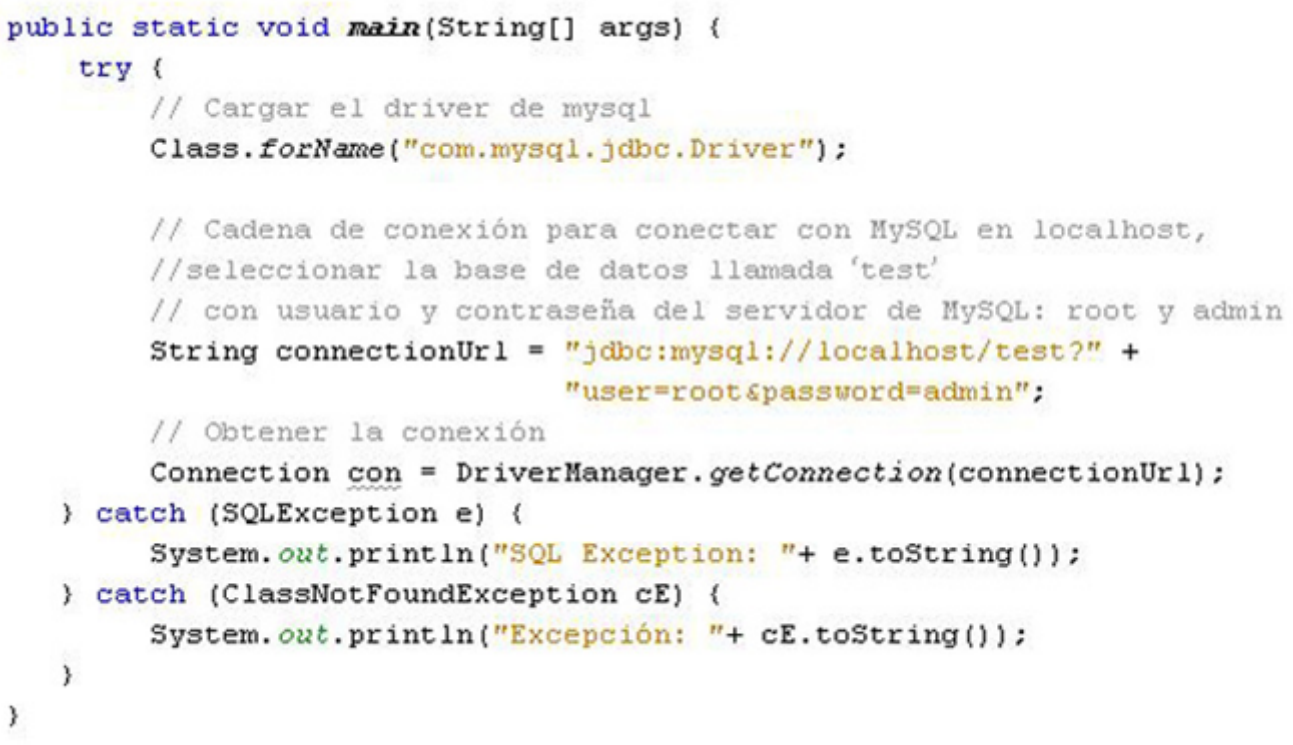
Cuando queremos acceder a una base de datos para operar con ella, lo primero que hay que hacer es conectarse a dicha base de datos.

En Java, para establecer una conexión con una base de datos podemos utilizar el método `getConnection()` de la clase `DriverManager`. Este método recibe como parámetro la URL de JDBC que identifica a la base de datos con la que queremos realizar la conexión.

La ejecución de este método devuelve un objeto `Connection` que representa la conexión con la base de datos.

Cuando se presenta con una URL específica, `DriverManager` itera sobre la colección de drivers registrados hasta que uno de ellos reconoce la URL especificada. Si no se encuentra ningún driver adecuado, se lanza una `SQLException`.

Veamos un ejemplo comentado:



```
public static void main(String[] args) {
    try {
        // Cargar el driver de mysql
        Class.forName("com.mysql.jdbc.Driver");

        // Cadena de conexión para conectar con MySQL en localhost,
        //seleccionar la base de datos llamada 'test'
        // con usuario y contraseña del servidor de MySQL: root y admin
        String connectionUrl = "jdbc:mysql://localhost/test?" +
                               "user=root&password=admin";
        // Obtener la conexión
        Connection con = DriverManager.getConnection(connectionUrl);
    } catch (SQLException e) {
        System.out.println("SQL Exception: "+ e.toString());
    } catch (ClassNotFoundException cE) {
        System.out.println("Excepción: "+ cE.toString());
    }
}
```

El mismo código copiable:

```
public static void main(String[] args) {
        try {
// Cargar el driver de mysql
                Class.forName("com.mysql.jdbc.Driver");
// Cadena de conexión para conectar con MySQL en localhost,
//seleccionar la base de datos llamada "test"
// con usuario y contraseña del servidor de MySQL: root y admin
String connectionUrl = "jdbc:mysql://localhost/test?" +
                                    "user=root&password=admin";
// Obtener la conexión
Connection con = DriverManager.getConnection(connectionUrl);
```

```
        } catch (SQLException e) {
```

```
            System.out.println("SQL Exception: "+ e.toString());
```

```
        } catch (ClassNotFoundException cE) {
```

```
            System.out.println("Excepción: "+ cE.toString());
```

```
         }
```

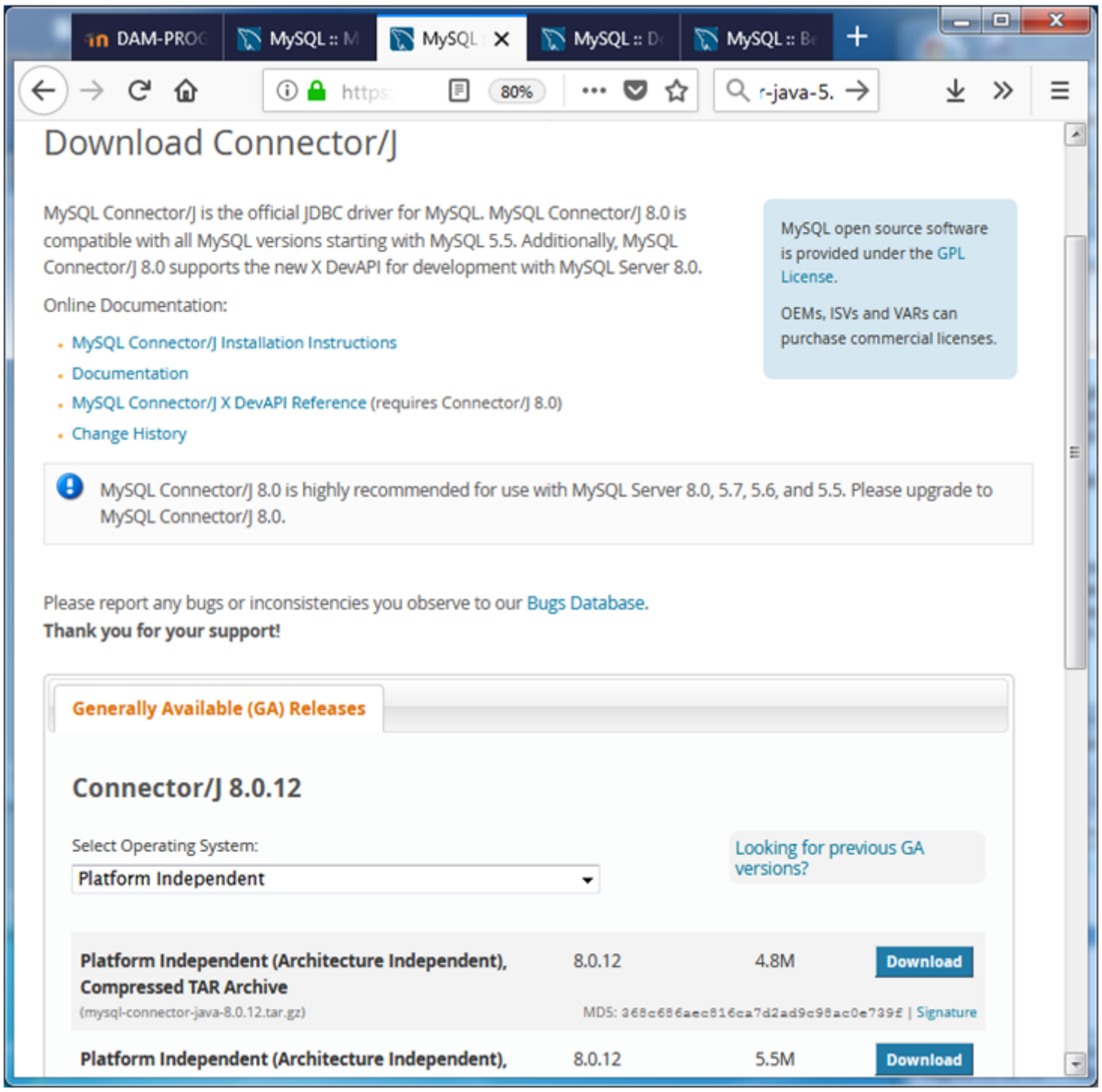
```
    }
```

Si probamos este ejemplo con Eclipse, NetBeans, o cualquier otro entorno, y no hemos instalado el conector para MySQL, en la consola obtendremos el mensaje: Excepción: `java.lang.ClassNotFoundException: com.mysql.jdbc.Driver.`

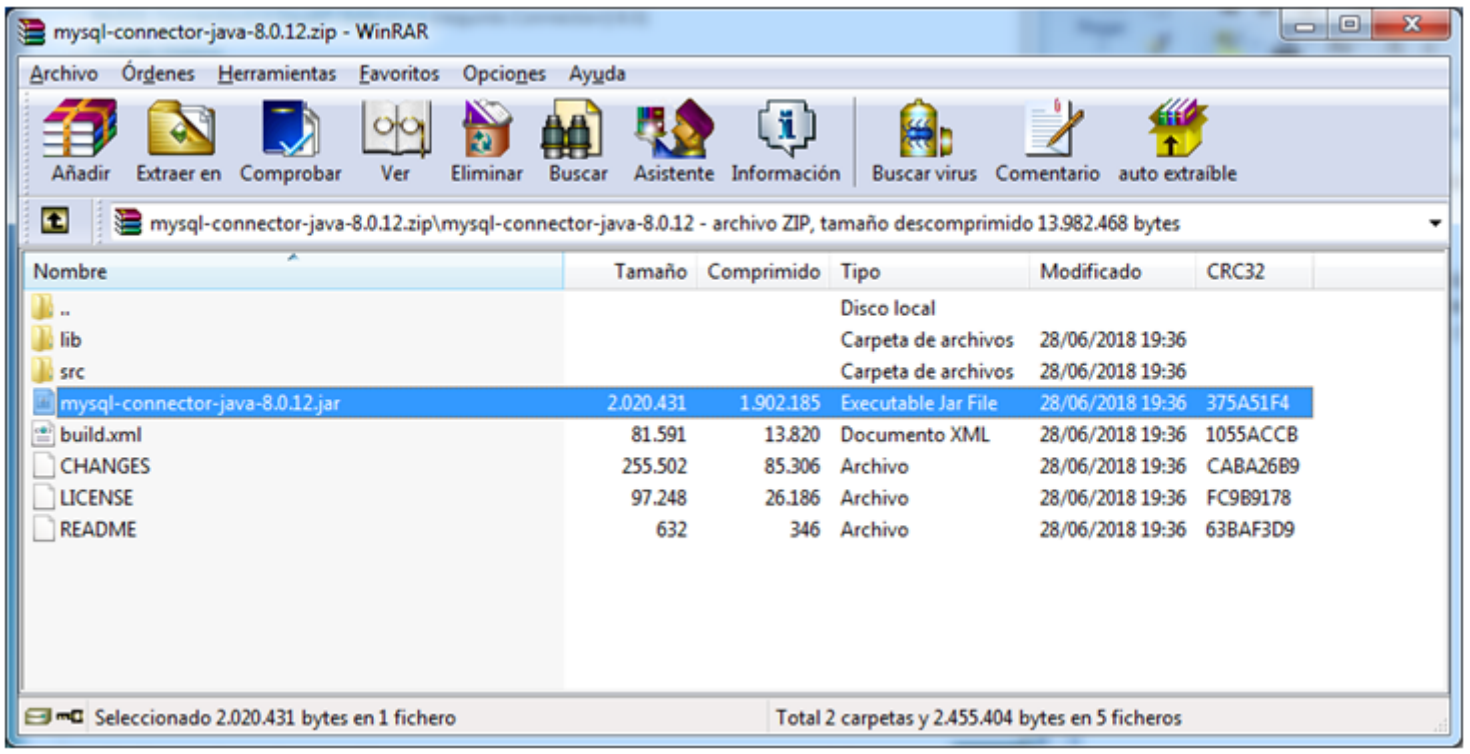
2. Instalar el conector de la base de datos.

En la siguiente explicación vamos a ver cómo descargarnos el conector o driver que necesitamos para trabajar con MySQL. Como verás, tan sólo consiste en descargar un archivo, descomprimirlo y desde Eclipse o NetBeans añadir el fichero `.jar` que constituye el conector que necesitamos.

De manera general accedemos a la dirección: http://www.mysql.com/downloads/connector/j/



En dicha página, en la parte de abajo, en el desplegable "Select Operating System:", elegiremos "Platform Independent", apareciendo dos posibles descargas (fichero TAR o fichero ZIP). Podemos elegir cualquiera de ellas, ya que descargaremos dicho fichero, y a continuación extraeremos un fichero .tar que se encuentra en cualquiera de ellos. Se trata del fichero **mysql-connector-java-8.0.12.jar**:



Una vez descargado y descomprimido el archivo, nos fijamos en el fichero que nos interesa (el .jar anteriormente indicado).

En NetBeans, situándonos en el nombre el proyecto pulsamos el botón derecho del ratón.

En el menú contextual que aparece seleccionamos Properties.

Seleccionamos el nodo de las Librerías del proyecto.

Pinchamos en el botón Add JAR/Folder.

Buscamos y elegimos el fichero comentado anteriormente, el .jar.

Tan solo queda pulsar Ok y hemos acabado.

Ejercicio propuesto

Haz la instalación del conector a la base de datos en el entorno Eclipse.

Por tanto, como ya hemos comentado anteriormente, entre el programa Java y el Sistema Gestor de la Base de Datos (SGBD) se intercala el conector JDBC. Este conector es el que implementa la funcionalidad de las clases de acceso a datos y proporciona la comunicación entre el API JDBC y el SGBD.

La función del conector es traducir los comandos del API JDBC al protocolo nativo del SGBD.

Autoevaluación

Para establecer una conexión con una base de datos se puede usar getConnection()**.**

- Verdadero.
- Falso.

3. Registrar el controlador JDBC.

Al fin y al cabo ya lo hemos visto en el ejemplo de código que poníamos antes, pero incidimos de nuevo. Registrar el controlador que queremos utilizar es tan fácil como escribir una línea de código.

Hay que consultar la documentación del controlador que vamos a utilizar para conocer el nombre de la clase que hay que emplear. En el caso del controlador para MySQL es "`com.mysql.jdbc.Driver`", o sea, que se trata de la clase **Driver** que está en el paquete `com.mysql.jdbc` del conector que hemos descargado, y que has observado que no es más que una librería empaquetada en un fichero .jar.

La línea de código necesaria en este caso, en la aplicación Java que estemos construyendo es:

```
// Cargar el driver de mysql
```

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

Una vez cargado el controlador, es posible hacer una conexión al SGBD.

Hay que asegurarse de que si no utilizáramos NetBeans u otro IDE, para añadir el .jar como hemos visto, entonces el archivo .jar que contiene el controlador JDBC para el SGBD habría que incluirlo en el CLASSPATH que emplea nuestra máquina virtual, o bien en el directorio ext del JRE de nuestra instalación del JDK.

Hay una excepción en la que no hace falta ni hacer eso: en caso de utilizar un acceso mediante puente JDBC-ODBC, ya que ese driver está incorporado dentro de la distribución de Java, por lo que no es necesario incorporarlo explícitamente en el classpath de una aplicación Java. Por ejemplo, sería el caso de acceder a una base de datos Microsoft Access.