JDBC

JDBC es la librería que incluye Java para trabajar directamente sobre cualquier base de datos. Utiliza las siguientes clases e interfaces:

- **DriverManager:** Es una clase que gestiona los modelos de bases de datos a los que nos podemos conectar (Oracle, MySQL, etc). Si queremos conectarnos con la base de datos de un fabricante, deberemos descargarnos su "driver", que no es más que un archivo JAR que nos permite conectarnos a un determinado modelo de base de datos. Dicho archivo JAR deberemos añadirlo a las librerías del proyecto en el IDE.

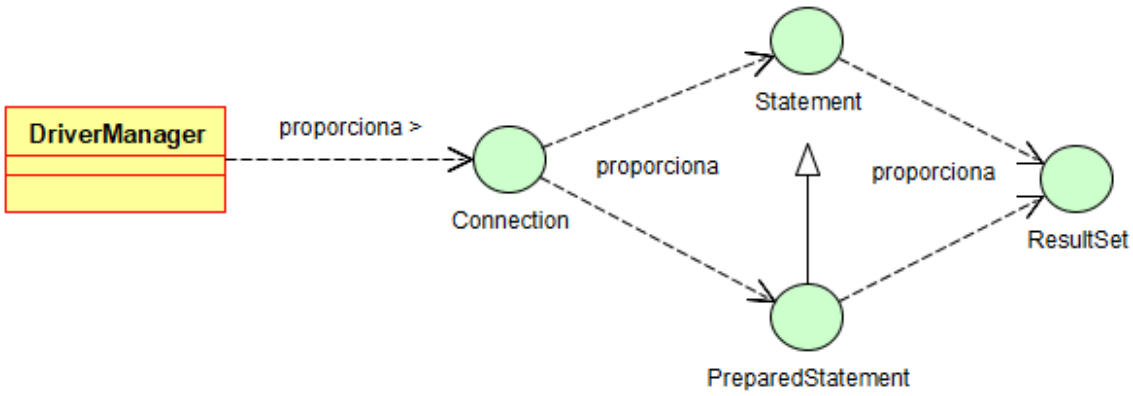
  Por defecto Java no incluye ningún driver, así que deberemos acudir a las páginas de los fabricantes para descargarnos sus drivers. Por ejemplo, para poder conectarnos con MySQL necesitamos descargarnos el driver JDBC de MySQL, que se encuentra en su página oficial.

- **Connection:** Es un objeto que representa una conexión activa sobre una base de datos. La clase DriverManager nos permite obtener objetos Connection. Una vez terminado el trabajo sobre la base de datos, deberemos cerrar la conexión para liberar recursos.

- **Statement:** Es un objeto que es capaz de lanzar sobre la base de datos una sentencia SQL escrita en un String. Sirve por tanto, para hacer consultas, inserciones, etc.

- **PreparedStatement:** Es una hija de Statement que nos facilita el trabajo cuando tenemos un String con una sentencia SQL en la que hay partes que deben ser sustituidos por variables del programa. Aunque no es obligatorio su uso, facilita las cosas.

- **ResultSet:** Representa el resultado de una consulta y tiene forma de tabla. Está especialmente diseñado para ser recorrido mediante un bucle while en el que se va bajando una por una por todas sus filas. Dentro de cada fila hay métodos que nos devuelven el valor de la columna que queramos, según su tipo de dato.



1) DriverManager

- **Librería:** *JDBC*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.sql*

DriverManager es la clase de JDBC que gestiona los drivers que tenemos cargados en un programa Java y además nos permite abrir una conexión con una base de datos concreta. Recordamos que un driver es una librería proporcionada por el fabricante de una base de datos para que JDBC sepa reconocerla y trabajar con ella.

DriverManager
+ static Connection getConnection(String cadenaConexión, String usuario, String password) throws SQLException + static Connection getConnection(String cadenaConexión) throws SQLException

- El primer método getConnection abre una conexión con la base de datos cuyas características pasamos como parámetro:

  - cadenaConexión: Es un String que identifica de forma única la base de datos a la que nos vamos a conectar. Cada modelo de base de datos tiene su propio formato de cadena de conexión, por lo que deberemos mirar las instrucciones proporcionadas por el fabricante antes de usar este método. En la última página se pueden ver los formatos de las cadenas de conexión de los principales SGBD. Por ejemplo, el formato usado por MySQL es así:

    jdbc:mysql://*servidor*:*puerto*/*nombreBD*

    Donde pone "servidor" pondríamos la IP o nombre del servidor. El "puerto" normalmente es 3306 para MySQL. El "nombre de la base de datos" es el nombre del esquema donde hemos creado las tablas. Un ejemplo concreto podría ser: jdbc:mysql://192.168.1.5:3306/FUTBOL

  - Usuario: String con el nombre de usuario que el administrador de la base de datos nos ha dado para conectarnos a la base de datos.

  - Contraseña: String con la contraseña de acceso correspondiente al usuario que usamos para conectarnos a la base de datos.

- El segundo método getConnection abre una conexión con la base de datos cuya cadena de conexión pasamos como parámetro. Este método no usa ningún usuario ni contraseña, por lo que solo sirve para bases de datos en las que la identificación no sea necesaria.

2) Connection

- **Librería:** *JDBC*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.sql*

Representa una conexión a una base de datos cuyas características hemos indicado en la cadena de conexión usada para llamar al método getConnection de la clase DriverManager. La clase Connection permite obtener información sobre la base de datos y obtener objetos para trabajar con la base de datos. También sirve para cerrar la conexión con la base de datos.

Connection
+ Statement createStatement() throws SQLException + PreparedStatement prepareStatement(String sql) throws SQLException + CallableStatement prepareCall(String sql) throws SQLException + void setAutoCommit(boolean b) throws SQLException + void commit() throws SQLException + void rollback() throws SQLException + close() throws SQLException

- createStatement devuelve un objeto con el que se pueden realizar operaciones de inserción, modificación y borrado.
- prepareStatement devuelve un objeto con el que se pueden realizar operaciones parametrizadas de inserción, modificación y borrado. Su uso es más sencillo para el programador de aplicaciones, aunque internamente funciona más lento que el createStatement.
- prepareCall devuelve un objeto con el que se pueden llamar a procedimientos almacenados en la base de datos.
- setAutoCommit activa (valor false) o desactiva (valor true) el modo transaccional sobre la base de datos.
- commit confirma la transacción activa y hace permanentes sus efectos sobre la base de datos.
- rollback deshace la transacción activa y retrotrae el estado de la base de datos al momento previo al inicio de la transacción.
- close cierra la conexión con la base de datos y libera sus recursos asociados.

3) Statement

- **Librería:** *JDBC*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.sql*

Esta clase define un objeto con el que es posible lanzar una sentencia escrita en lenguaje SQL sobre la base de datos.

Statement
+ ResultSet executeQuery(String sql) throws SQLException + int executeUpdate(String sql) throws SQLException + Connection getConnection() throws SQLException + void close() throws SQLException

- El método executeQuery permite lanzar un String que tenga escrita una consulta (SELECT) a la base de datos y devuelve un objeto con la tabla resultado de la consulta.
- El método executeUpdate permite lanzar un String rellenado con una sentencia SQL correspondiente a una inserción (INSERT), modificación (UPDATE) o borrado (DELETE) sobre la base de datos. Devuelve el número de filas afectadas por dicha operación.
- El método getConnection devuelve la conexión sobre la que se ha creado el statement
- El método close cierra los recursos asociados al statement. Normalmente los statements se cierran solos al cerrar la conexión, por lo que en un uso normal no se necesita tener que llamar a este método. Sin embargo, hay veces (por ejemplo, cuando se usan varios statements a la vez como resultado de consultas anidadas) en los que hay que cerrarlos manualmente con este método.

4) PreparedStatement

- **Librería:** *JDBC*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.sql*

Esta clase define un objeto con el que es posible lanzar una sentencia escrita en lenguaje SQL sobre la base de datos. Los valores de ese String que deban corresponder con datos del programa que estén guardados en variables se dejarán con el signo de interrogación ?. La clase tiene métodos para cambiar cada interrogante por el valor de una variable.

PreparedStatement
+ ResultSet executeQuery()throws SQLException + int executeUpdate ()throws SQLException + void setString (int pos, String s) throws SQLException + void setInt (int pos, int n) throws SQLException + void setBoolean (int pos, boolean b) throws SQLException + void setLong (int pos, long l) throws SQLException + void setDouble(int pos, double d) throws SQLException + void setDate(int pos, Date d) throws SQLException + void setBinaryStream (int pos, OutputStream a) throws SQLException + void clearParameters() throws SQLException

- El método executeQuery permite lanzar la consulta (SELECT) a la base de datos y devuelve un objeto con la tabla resultado de la consulta.
- El método executeUpdate permite lanzar la operación de inserción (INSERT), modificación (UPDATE) o borrado (DELETE) sobre la base de datos. Devuelve el número de filas afectadas por dicha operación.
- El método setString permite cambiar el ? de la posición indicada en el primer parámetro por un String. El primer ? es el 1.
- El método setInt permite cambiar el ? de la posición indicada en el primer parámetro por un número entero. El primer ? es el 1.
- El método setBoolean permite cambiar el ? de la posición indicada en el primer parámetro por un booleano. El primer ? es el 1.
- El método setLong permite cambiar el ? de la posición indicada en el primer parámetro por un long. El primer ? es el 1.
- El método setDouble permite cambiar el ? de la posición indicada en el primer parámetro por un double. El primer ? es el 1.
- El método setDate permite cambiar el ? de la posición indicada en el primer parámetro por un Date. El primer ? es el 1.
- El método setBinaryStream permite añadir a la columna indicada en la posición del ?, el flujo de datos pasado como parámetro
- El método clearParameters borra los valores establecidos con los métodos "set" y deja el objeto como al principio, con todos sus ? intactos.

5) ResultSet

- **Librería:** *JDBC*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.sql*

El ResultSet representa la tabla resultado de lanzar una sentencia SQL a una base de datos. Internamente el ResultSet tiene forma de tabla en la que hay una flecha que señala a una fila. Inicialmente, dicha flecha se encuentra fuera de la tabla, justo por encima de la primera fila.

ResultSet
+ boolean next() throws SQLException + String getString (int pos) throws SQLException + int getInt (int pos) throws SQLException + boolean getBoolean (int pos) throws SQLException + long getLong (int pos) throws SQLException + double getDouble(int pos) throws SQLException + Date getDate(int pos) throws SQLException + InputStream getBinaryStream (int pos) throws SQLException + boolean wasNull() throws SQLException

- El método next hace que baje la flecha del ResultSet y ésta apunte a la siguiente fila. Devuelve false si la flecha se sale de la tabla por abajo.
- getString devuelve un String con el contenido de la columna de tipo texto cuya posición se pasa como parámetro. La numeración empieza en 1.
- getInt devuelve un entero con el contenido de la columna de tipo entero cuya posición se pasa como parámetro. La numeración empieza en 1.
- getBoolean devuelve un boolean con el contenido de la columna de tipo boolean cuya posición se pasa como parámetro. La numeración empieza en 1.
- getLong devuelve un long con el contenido de la columna de tipo entero largo cuya posición se pasa como parámetro. La numeración empieza en 1.
- getDouble devuelve un double con el contenido de la columna de tipo número real cuya posición se pasa como parámetro. La numeración empieza en 1.
- getDate devuelve un Date con el contenido de la columna de tipo fecha/hora cuya posición se pasa como parámetro. La numeración empieza en 1.
- getBinaryStream devuelve un canal de entrada de bytes para leer el contenido de la columna cuya posición se pasa como parámetro. La numeración empieza en 1.
- wasNull devuelve true si la última columna leída con un método getX ha devuelto el valor nulo (NULL) de la base de datos. Se utiliza para detectar valores nulos en las columnas de la base de datos.

6) CallableStatement

- **Librería:** *JDBC*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.sql*

Esta clase nos permite realizar todo lo que puede hacerse con un PreparedStatement y además, llamar a funciones y procedimientos almacenados en una base de datos. Los objetos de esta clase se obtienen mediante el método prepareCall de la conexión, y deberemos pasar a este método un String con el siguiente formato, según lo que queramos invocar:

- Procedimiento almacenado: **{call** *nombre_del_procedimiento***(?,...,?)}**
- Función almacenada:**{?=call** *nombre_de_la_función***(?,...,?)}**

En ambos casos se pone un ? por cada parámetro de la función o procedimiento. Para asignar el valor de los ?, se utilizan los métodos **set** que heredamos de PreparedStatement.

CallableStatement extends PreparedStatement
+ void execute() throws SQLException + void registerOutParameter(int número, int tipo) throws SQLException + String getString (int pos) throws SQLException + int getInt (int pos) throws SQLException + boolean getBoolean (int pos) throws SQLException + long getLong (int pos) throws SQLException + double getDouble(int pos) throws SQLException + Date getDate(int pos) throws SQLException + InputStream getBinaryStream (int pos) throws SQLException + boolean wasNull() throws SQLException

- El método execute sirve para lanzar un procedimiento almacenado.
- El método registerOutParameter sirve para asociar un ? a un valor devuelto por una función almacenada. El primer parámetro es el número de interrogante y el segundo es una constante que indica el tipo de dato de dicho parámetro:

Types.INTEGER	Número entero
Types.VARCHAR	Texto de longitud variable
Types.BOOLEAN	Valor booleano
Types.CHAR	Un solo carácter
Types.DECIMAL	Un número con decimales
Types.DATE	Una fecha

- Los métodos get devuelve el valor de un ? que ha sido previamente configurado como un parámetro de salida.
- El método wasNull devuelve true si la última llamada a un método get ha devuelto un valor nulo.

Drivers y Cadenas de conexión

- **Archivo del driver:** ojdbc14.jar
- **Clase del driver:** oracle.jdbc.driver.OracleDriver
- **Cadena de conexión:** jdbc:oracle:thin:@*servidor*:*puerto*:*nombreBD*

- **Archivo del driver:** mysql-connector-java-5.0.4-bin.jar
- **Clase del driver:** com.mysql.jdbc.Driver
- **Cadena de conexión:** jdbc:mysql://*servidor*:*puerto*/*nombreBD*

- **Archivos del driver:** derby.jar, derbyclient.jar, derbyLocale_es.jar
- **Clase del driver:** org.apache.derby.jdbc.EmbeddedDriver
- **Cadena de conexión:**jdbc:derby:*directorioBD*

- **Archivos del driver open source jTDS para Windows:** jTDS-1.2.5.jar, ntlmauth.dll
- **Clase del driver:** net.sourceforge.jtds.jdbc.Driver
- **Cadena de conexión:**jdbc:jtds:sqlserver://*servidor*:*puerto*/*nombreBD*

- **Archivo del driver:** postgresql-9.1-901.jdbc3.jar
- **Clase del driver:** org.postgresql.Driver
- **Cadena de conexión:** jdbc:postgresql://*servidor*:*puerto*/*nombreBD*

¿Para qué sirve la "clase del driver"?: Para drivers muy antiguos, es necesario usar el método **Class.forName(*clase del driver*)** para cargar el driver. En los drivers más recientes (como los que usamos hoy) esto no hace falta.