Tema BDRR: Manejo de conectores

Conector: Conjunto de clases encargadas de implementar la API para facilitar el acceso a una BD (interfaz), permitiendo el desarrollo de una única aplicación independientemente del SGBD al que nos conectemos

Protocolos de acceso a BD - Tipos de conectores:

- ODBC (Open Database Connectivity): API desarrollada por Microsoft que permite a una aplicación (de cualquier lenguaje) conectarse y trabajar con una BD
- JDBC (Java Database Connectivity): API desarrollada por Oracle que permite a una aplicación Java conectarse y trabajar con una BD relacional. Además de otorgarnos la interfaz define una arquitectura estándar de AD, existen varias interfaces/drivers: MySQL, SQLite, ApacheDerby, HSQLDB, H2, DB4O, Oracle
- JDBC-ODBC: Permite convertir llamadas JDBC en ODBC para acceder a BD que poseen un conector ODBC, pero no uno JDBC

MySQL	ApacheDerby
`String driver = "com.mysql.jdbc.Driver";`	`String driver = "org.apache.derby.jdbc.EmbeddedDriver";`
`String url = "jdbc:mysql://localhost:3307?user=root&password=usbw";`	`String url = "jdbc:derby:bd/ApacheDerby/ejemplo;create=true";`

Tipos de drivers:

1. JDBC-ODBC Bridge: Exige la instalación y configuración ODBC en el cliente
2. Native: Escrito parcialmente en Java y lenguaje nativo, traduce las sentencias de JDBC a las del SGBD. Requiere instalar código binario propio del cliente de BD en el cliente
3. Network: Controlador Java puro que utiliza un protocolo de red para conectarse al SGBD. Traduce las sentencias de JDBC a las propias del protocolo de red. No requiere instalación en el cliente
4. Thin: Controlador Java puro con protocolo nativo. Traduce las sentencias JDBC en las usadas por el protocolo de red del SGBD. No requiere instalación en el cliente

Los tipos 1 y 2 solo se usarán en caso de extrema necesidad, es decir si al SGBD solo se puede acceder mediante ODBC, en la mayoría de los casos la opción más usada es la de tipo 4

Interfaces de JDBC:

Todas ellas están definidas en el paquete java.sql, las más importantes son:

- DirverManager: Permite gestionar los driver instalados en el sistema
- Connection: Representa una conexión con una BD, en una misma aplicación podemos tener varias
- DatabaseMetadata: Proporciona información acerca de una BD
- Statement: Permite ejecutar sentencias SQL sin parámetros de entrada
- ResultSet: Contiene las tuplas resultantes de una sentencia SELECT

Arquitecturas JDBC:

- Modelo de dos capas: Un applet/aplicación Java se comunica directamente con la BD, requiere un driver JDBC en la aplicación que será el que gestione la comunicación
  - La aplicación envía sentencias SQL al SGBD y este las procesa y envía los resultados de vuelta
- Modelo de tres capas: Las sentencias SQL se envían a una capa intermedia que enviará los comandos a la BD y recogerá los resultados a la aplicación, en este caso el conector JDBC no está en la misma máquina que la aplicación

Desfase Objeto-Relacional: Se denomina al problema generado al intentar gestionar BD relacionales mediante lenguajes orientados a objetos

Funcionamiento de una aplicación con JDBC:

Siempre al principio del programa	Para una consulta
Importamos las clases necesarias (línea 19)	Ejecutar una consulta con el objeto Statement
Cargamos el origen de datos (String URL)	Recuperar los datos del objeto ResultSet
Creamos un objeto Connection (línea 21)	Liberar el objeto ResultSet
Creamos un objeto Statement (línea 23)	Regencias al método conectarBD siguiente pagina
Al finalizar el programa (automático)	
Liberar el objeto Statement	
Liberar el objeto Connection	

Acceso a BD relacional mediante JDBC:

Para ello necesitaremos las siguientes variables:

```
VARIABLES GLOBALES
static Connection conexion;
static Statement sentencia;
static ResultSet resultado;
```

```
VARIABLES LOCALES
String driver = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql://localhost:3307?user=root&password=usbw";
String nombreBD = "bdEjemplo";
```

Antes de entrar al menú conectamos con el driver JDBC y creamos si es necesario la BD

```java
@SuppressWarnings("UseSpecificCatch")
public static void conectarBD (String url, String driver, String nombreDB) {

    try {

        //cargamos el driver
        Class.forName(driver);
        //establecemos la conexion
        conexion = DriverManager.getConnection(url);
        //creamos la sentencia
        sentencia = conexion.createStatement();
    } catch (ClassNotFoundException e) {

        System.err.println(" No se encontro el driver " + driver);
        System.exit(0);
    } catch (SQLException e) {

        System.err.println(" No se establecio la conexion ");
        System.exit(0);
    } catch (Exception e) {

        System.err.println(" Se produjo un error al conectarse con el driver que maneja la BD " +
                           " u otro error indeterminado ");
        System.exit(0);
    }
    Crear.estructuraBD(sentencia, resultado, nombreDB);
}
```

```java
public static void estructuraBD (Statement sentencia, ResultSet resultado, String nombreDB) {

    boolean bdExists = false;

    try {

        resultado = sentencia.executeQuery("SHOW DATABASES;");
        while (resultado.next()) {

            if (resultado.getString("Database").compareToIgnoreCase(nombreDB) == 0) {

                bdExists = true;
            }
        }
        resultado.close();

        if (!bdExists) {

            System.out.println("\n La BD " + nombreDB + " y sus tablas no existe se procede a su creacion \n");

            sentencia.execute("CREATE DATABASE IF NOT EXISTS " + nombreDB + ";");
            sentencia.execute("USE " + nombreDB + ";");
            System.out.println("\n La BD " + nombreDB + " se ha creado exitosamente ");
            //CREACION DE TABLAS
            sentencia.execute("CREATE TABLE IF NOT EXISTS tablaB ( " +
                    "   idB INT(4) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT, " +
                    "   nombre VARCHAR, " +
                    "   idA INT(4) UNSIGNED ZEROFILL NOT NULL, " +
                    "   PRIMARY KEY (idB), " +
                    "   CONSTRAINT fk1_tablaA " +
                    "       FOREIGN KEY(idA) " +
                    "       REFERENCES tablaA (idA) " +
                    "       ON DELETE CASCADE " +
                    "       ON UPDATE CASCADE, " +
                    "   INDEX fk1_productos (idA) " +
                    ");");
            System.out.println("\n La tabla tablaA se ha creado exitosamente en la BD " + nombreDB);
        } else {

            System.out.println("\n La BD " + nombreDB + " y sus tablas ya existen \n");
        }
    } catch (SQLException e) {

        System.err.println(" Sentencia SQL no ejecutada - Error durante la conexion a la BD y/o " +
                           "creacion de la estructura BD ");
    }
}
```

Para las inserciones y modificaciones usaremos el método sentencia.excuteUpdate("sql") y para las consultas el método sentencia.executeQuery("sql")

Siempre que realicemos una consulta si solo obtenemos un resultado usaremos el método resultado.first(), si por el contrario recibimos una lista la iteramos mediante while(resultado.next())

Cada vez que trabajemos con un objeto de la clase ResultSet debemos ejecutar resultado.close() para liberar el mismo