GOBIERNO DE LA CIUDAD DE MÉXICO

AGENCIA DIGITAL DE INNOVACIÓN PÚBLICA

PILARES

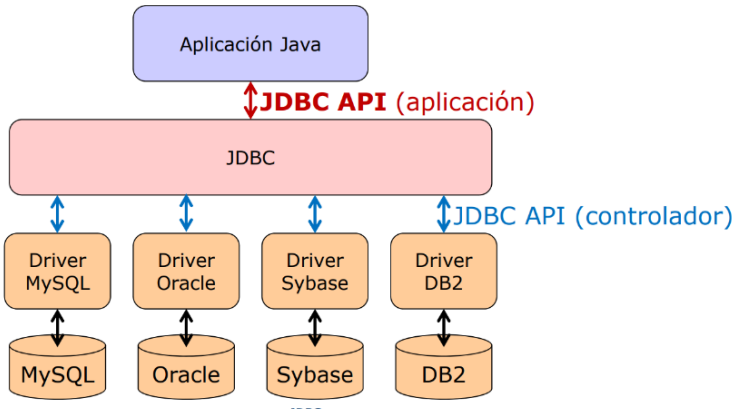
Programador Junior

Proyecto Final:

Manual del JDBC

MANUAL DE JDBC(Java Database Connectivity)

JDBC (Java Database Connectivity) es un API de Java que nos permite conectarnos con bases de datos y realizar operaciones sobre ellas utilizando instrucciones SQL desde una aplicación Java. Con JDBC tenemos una interfaz para conectarnos con una base de datos sin tener que preocuparnos de si es una base de datos MySQl, Oracle, SQLServer o cualquier otro tipo de base de datos. El único cambio que habría que hacer para cambiar el tipo de base de datos de una aplicación sería cambiar el driver específico de la base de datos en cuestión.



El paquete actual de JDK incluye JDBC y el puente JDBC-ODBC. Estos paquetes son para su uso con JDK 1.0

Para usar JDBC hay que seguir los siguientes pasos:

1. INCLUIR EL JAR CON EL DRIVER DE LA BASE DE DATOS

El primer paso es obtener el driver de la base de datos que vamos a utilizar, buscamos en google «MySQL jdbc driver», «Oracle jdbc driver» o el que queramos y descargamos el jar y lo incluimos es nuestro proyecto. Para este ejemplo voy a usar MySQL, puedes descargarlo desde aquí, lo que queremos es el .jar por lo que tienes que elegir Plataform Independent y descargar el zip o el tar.gz, y aunque vienen unos cuantos archivos y carpetas el único que nos interesa es mysql-connector-java-5.1.26-bin.jar que lo tenemos que añadir a nuestro proyecto.

2. CARGAR EL DRIVER

Ya tenemos el jar con el driver, pero hay que cargarlo para que se pueda hacer uso de él en nuestra aplicación. En nuestro caso como vamos a usar MySQL la instrucción es la siguiente y como puede lanzar una excepción pues se mete dentro de un try-catch.

```
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException ex) {
    log.error("No se encontro el Driver MySQL para JDBC.");
}
```

3.ESTABLECER UNA CONEXIÓN CON LA BASE DE DATOS

El siguiente paso es establecer una conexión con la base de datos. Se crea una nueva conexión con la base de datos.

```
//Connection cn = DriverManager.getConnection("jdbc:mysql://servidor_bd:puerto/nombre_bd", "usuario",
Connection cn = DriverManager.getConnection("jdbc:mysql://localhost:3306/cuentas", "root", "");
```

Para obtener una conexión tenemos que hacer uso del método getConnection de DriverManager y le tenemos que pasar como parámetros, la base de datos, el usuario y la contraseña. Si en lugar de usar mysql se usa otra base de datos el formato de la url de la base de datos cambia, por ejemplo para oracle el formato es jdbc:oracle:<tipo_driver>:@<<base_datos>.

4. REALIZAR UNA CONSULTA

Una vez obtenida la conexión ya podemos hacer consultas a la base de datos. Hay tres métodos para ejecutar una consulta de tipo Statement que es una consulta normal, PreparedStatement con la que se puede crear una consulta que se pre compila en la base de datos y como es de esperar se pueden establecer los distintos datos porque aunque la estructura de la consulta será la misma lo lógico es que los datos de la consulta sean distintos y finalmente están las CallableStatements que sirven para ejecutar procedimientos almacenados.

Vamos a ver como hacer una consulta de tipo Statement:

```
// Creamos el Statement para poder hacer consultas
Statement st = cn.createStatement();

// La consulta es un String con código SQL
String sql1 = "SELECT * FROM cuentas";

// Ejecuta una consulta que devuelve resultados
ResultSet rs = st.executeQuery(sql1);
while (rs.next()) {
    System.out.println (rs.getString ("propietario") + " " + rs.getString (2)+ " " + rs.getInt(saldo));
}

String sql2 = "UPDATE cuentas SET saldo = saldo - "
                    + cantidad + " WHERE codigo = '"
                    + codigo + "' AND saldo >= "
                    + cantidad;

// Ejecuta una consulta de tipo insert, update o delete
st.executeUpdate(sql2);
```

Crear un statement sobre el que luego podemos hacer las consultas que queramos una vez creado para hacer nuestra consulta tenemos los métodos execute (para hacer cualquier consulta, pero su valor de devolución es boolean y devuelve true si lo que devuelve es un resulset y falso en caso contrario) exeteQuery (para hacer select) y executeUpdate (para hacer insert, update y delete). Y finalmente si se ejecuta una query podemos obtener el resultado mediante ResultSet, para obtener los valores de las columnas resulset tiene un get para cada tipo de datos que se puede llamar pasándole como parámetro el nombre de la columna de base de datos o el número de la columna según se prefiera.

Si usamos **PreparedStatement** además de que son más eficientes nos facilitan la tarea de escribir las consultas SQL porque en lugar de tener que estar concatenando las distintas variables con el código SQL, lo que se hace es sustituir en este las variables por ? y posteriormente se introducen los datos concretos mediante setters.

```
PreparedStatement pstBuscarCodigo;
PreparedStatement pstInsertarCuenta;

String sqlBusqueda = "SELECT codigo FROM cuentas WHERE codigo=?";
pstBuscarCodigo = cn.prepareStatement(sqlBusqueda);
pstBuscarCodigo.setString(1, codigo);
ResultSet rs = pstBuscarCodigo.executeQuery();
if (!rs.next){...}

String sqlNuevaCuenta = "INSERT INTO cuentas VALUES (?,?,?,?)";
pstInsertarCuenta = cn.prepareStatement(sqlNuevaCuenta);
pstInsertarCuenta.setString(1, codigo);
pstInsertarCuenta.setString(2, nombre);
pstInsertarCuenta.setString(3, email);
pstInsertarCuenta.setDouble(4, saldo);
pstInsertarCuenta.executeUpdate();
```

5. CERRAR LA CONEXIÓN

Después de hacer las consultas que se necesite se debe de cerrar la conexión para liberar los recursos, también se pueden cerrar el resulset y el statement de forma manual pero cerrando la conexión se cierran los otros dos porque están creados a partir de la conexión, del mismo modo al cerrar el statement también se liberan los recursos del resulset.

```
rs.close(); // Cierra el resulset
st.close(); // Cierra el statement
cn.close(); // Cierra la conexión
```