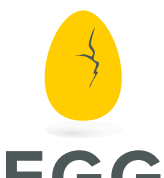
CURSO DE PROGRAMACIÓN FULL STACK

ACCESO A BASES DE DATOS DESDE JAVA: JDBC



GUÍA DE PERSISTENCIA CON JDBC Y JPA

¿QUE ES JDBC?

Java™ Database Connectivity (JDBC) es la especificación JavaSoft de una interfaz de programación de aplicaciones (API) estándar que permite que los programas Java accedan a sistemas de gestión de bases de datos. La API JDBC consiste en un conjunto de interfaces y clases escritas en el lenguaje de programación Java.

Con estas interfaces y clases estándar, los programadores pueden escribir aplicaciones que conecten con bases de datos, envíen consultas escritas en el lenguaje de consulta estructurada (SQL) y procesen los resultados.

Puesto que JDBC es una especificación estándar, un programa Java que utilice la API JDBC puede conectar con cualquier sistema de gestión de bases de datos (DBMS), siempre y cuando haya un driver para dicho DBMS en concreto.

COMPONENTES DE JDBC

En general, hay dos componentes principales de JDBC a través de los cuales puede interactuar con una base de datos. Son los que se mencionan a continuación:

JDBC Driver Manager: carga el driver específico de la base de datos en una aplicación para establecer una conexión con una base de datos. Se utiliza para realizar una llamada específica de la base de datos a la base de datos para procesar la solicitud del usuario.

API JDBC: Es un conjunto de interfaces y clases, que proporciona varios métodos e interfaces para una fácil comunicación con la base de datos. Proporciona dos paquetes de la siguiente manera que contiene las plataformas java SE y java EE para exhibir capacidades WORA (*write once run everything*).

Estos paquetes son:

1. java.sql.*;

2. javax.sql.*;

Las clases e interfaces principales de JDBC son:

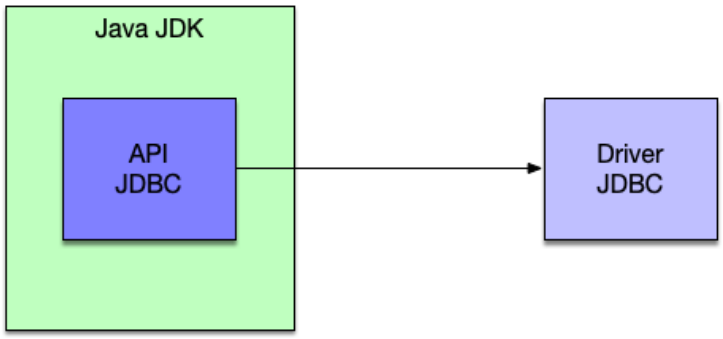
- java.sql.DriverManager
- java.sql.Connection
- java.sql.Statement
- java.sql.ResultSet
- java.sql.PreparedStatement
- javax.sql.DataSource

¿QUÉ ES UN DRIVER JDBC?

Vimos que dentro de los componentes de JDBC existe el Driver Manager que es el encargado de cargar el driver, pero que es el driver exactamente.

La API JDBC define las interfaces y clases Java™ que utilizan los programadores para conectar con bases de datos y enviar consultas. Un driver JDBC implementa dichas interfaces y clases para un determinado proveedor de DBMS.

Un programa Java que utiliza la API JDBC carga el controlador especificado para el DBMS particular antes de conectar realmente con una base de datos. Luego la clase JDBC DriverManager envía todas las llamadas de la API JDBC al controlador cargado.



Cada base de datos debe aportar sus propias implementaciones y es ahí donde el Driver JDBC realiza sus aportes. El concepto de Driver hace referencia al conjunto de clases necesarias que implementa de forma nativa el protocolo de comunicación con la base de datos en un caso será Oracle y en otro caso será MySQL.

Por lo tanto para cada base de datos deberemos elegir su Driver .¿Cómo se encarga Java de saber cual tenemos que usar en cada caso?. Muy sencillo, Java realiza esta operación en dos pasos. En el primero registra el driver con la instrucción:

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

Una vez registrado el Driver , este es seleccionado a través de la propia cadena de conexión que incluye la información sobre cual queremos usar, en la siguiente línea podemos ver que una vez especificado el tipo de conexión define el Driver “MySql”

```
String url="jdbc:mysql://localhost:3306/biblioteca";
```

COMPONENTES DEL API DE JDBC

- Driver: Es el enlace de comunicaciones de la base de datos que maneja toda la comunicación con la base de datos. Normalmente, una vez que se carga el controlador, el desarrollador no necesita llamarlo explícitamente.
- Connection: Es una interfaz con todos los métodos para contactar una base de datos. El objeto de conexión representa el contexto de comunicación, es decir, toda la comunicación con la base de datos es solo a través del objeto de Connection.
- Statement: Encapsula una instrucción SQL que se pasa a la base de datos para ser analizada, compilada, planificada y ejecutada.
- ResultSet: Los ResultSet representan un conjunto de filas recuperadas debido a la ejecución de una consulta.

ACCESO A BASES DE DATOS CON JDBC

JDBC nos permitirá acceder a bases de datos desde Java. Para ello necesitaremos contar con un SGBD (sistema gestor de bases de datos) además de un driver específico para poder acceder a este SGBD. La ventaja de JDBC es que nos permitirá acceder a cualquier tipo de base de datos, siempre que contemos con un driver apropiado para ella.

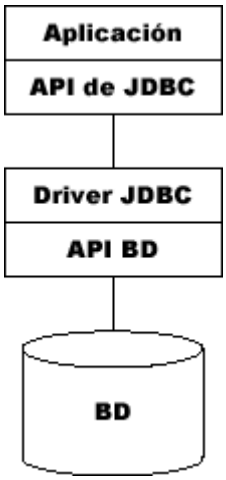


Figura 1: Arquitectura de JDBC

Como se observa en la Figura 1, cuando se construye una aplicación Java utilizando JDBC para el acceso a una base de datos, en la aplicación siempre se utiliza la API estándar de JDBC, y la implementación concreta de la base de datos será transparente para el usuario.

CONEXIÓN CON LA BASE DE DATOS

Para comunicar con una base de datos utilizando JDBC, se debe en primer lugar establecer una conexión con la base de datos a través del driver JDBC apropiado. El API JDBC especifica la conexión en la interfaz java.sql.Connection.

La clase DriverManager permite obtener objetos Connection con la base de datos.

Para conectarse es necesario proporcionar:

- URL de conexión, que incluye:
  - Nombre del host donde está la base de datos.
  - Nombre de la base de datos a usar.
- Nombre del usuario en la base de datos.
- Contraseña del usuario en la base de datos.

El siguiente código muestra un ejemplo de conexión y obtención de datos en JDBC a una base de datos MySQL:

```
Connection connection = null;
(…)
try {
      Class.forName("com.mysql.jdbc.Driver");
      String url = “jdbc:mysql://hostname/database-name”;
```

```
        connection = DriverManager.getConnection(url, “user”, “password”);
        }
} catch (SQLException ex) {
        connection = null;
        ex.printStackTrace();
        System.out.println(“SQLException: ” + ex.getMessage());
        System.out.println(“SQLState: ” + ex.getSQLState());
        System.out.println(“VendorError: ” + ex.getErrorCode());
}
```

En este ejemplo, primero se revisa el driver con la sentencia ClassForName. Después, la clase DriverManager intenta establecer una conexión con la base de datos database-name utilizando el driver JDBC que proporciona MySQL. Para poder acceder al RDBMS MySQL es necesario introducir un login y un password válidos. En el API JDBC, hay varios métodos que pueden lanzar la excepción SQLException.

Conexión a la Base de Datos (Objeto Connection)

Una vez cargado el driver apropiado para nuestro SGBD se debe establecer la conexión con la BD. Para ello se utiliza el siguiente método:

```
Connection con = DriverManager.getConnection(url, login, password);
```

La conexión a la BD está encapsulada en un objeto Connection, y para su creación se debe proporcionar la url de la BD y el login y password para acceder a ella. El formato de la url variará según el driver que se utilice.

El objeto Connection representa el contexto de una conexión con la base de datos, es decir:

- Permite obtener objetos Statement para realizar consultas SQL.
- Permite obtener metadatos acerca de la base de datos (nombres de tablas, etc.)
- Permite gestionar transacciones.

OBTENCIÓN DE DATOS DE LA BASE DE DATOS

Una vez que tengamos la conexión con el objeto Connection, la vamos a usar para crear un objeto Statement, este objeto recibe la consulta para ejecutarla y enviársela a la base de datos. La información que recibimos de la base de datos, va a ser capturada por el objeto ResulSet para después poder mostrar la información.

```
connection = DriverManager.getConnection(url, “user”, “password”);
String sql = "SELECT a, b, c FROM Table1";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
int x = rs.getInt("a");
String s = rs.getString("b");
double d = rs.getDouble("c");
System.out.println("Fila = " + i + " " + s + " " + f);
```

Creación y ejecución de sentencias SQL (Objeto Statement)

- Los objetos Statement permiten realizar consultas SQL en la base de datos.
- Se obtienen a partir de un objeto Connection.

Una vez obtenida la conexión a la BD, se puede utilizar para crear sentencias. Estas sentencias están encapsuladas en la clase Statement, y se pueden crear de la siguiente forma:

```
Statement stmt = con.createStatement();
```

Tienen distintos métodos para hacer consultas:

- executeQuery: para sentencias SQL que recuperen datos de un único objeto ResultSet. Es usado para leer datos (típicamente consultas SELECT).
- executeUpdate: para realizar actualizaciones que no devuelvan un ResultSet. Es usado para insertar, modificar o borrar datos (típicamente sentencias INSERT, UPDATE y DELETE).

Una vez obtenido este objeto se puede ejecutar sentencias utilizando su método executeUpdate() al que se proporciona una cadena con la sentencia SQL que se quiere ejecutar:

```
stmt.executeQuery(sentenciaSQL);
```

Estas sentencias pueden utilizarse para creación de tablas, y para la inserción, actualización y borrado de datos en ellas.

Las sentencias SQL van a ser las mismas que veníamos trabajando en MySQL Workbench, se van a poner entre comillas dobles ya que el objeto Statement recibe String, como dato para las sentencias.

```
String sentenciaSQL = "SELECT nombre, apellido FROM Alumnos";
```

Como podemos ver las sentencias van a tener la misma sintaxis que veníamos trabajando, la única diferencia, se va a presentar a la hora de trabajar con datos de tipo String y de tipo Date. Como estos datos suelen ir en comillas dobles en Java y en SQl, y nuestra sentencia ya está entre comillas dobles, deberemos poner los datos entre comillas simples para diferenciarlos.

String:

```
"SELECT nombre, apellido FROM Alumnos WHERE nombre = 'Agustin'";
```

Date:

```
"SELECT nombre FROM Alumnos WHERE fechaNacimiento = '01-11-1990'";
```

Obtención de datos (Objeto ResulSet)

Para obtener datos almacenados en la BD se utiliza una consulta SQL (query). La consulta se puede ejecutar utilizando el objeto Statement, con el método executeQuery() al que le se le pasa una cadena con la consulta SQL. Los datos resultantes se devuelven como un objeto ResultSet.

```
ResultSet result = stmt.executeQuery(query);
```

La consulta SQL devolverá una tabla, que tendrá una serie de campos y un conjunto de registros, cada uno de los cuales consistirá en una tupla de valores correspondientes a los campos de la tabla.

El objeto ResultSet proporciona el acceso a los datos de estas filas mediante un conjunto de métodos get que permiten el acceso a las diferentes columnas de la filas. El método ResultSet.next se usa para moverse a la siguiente fila del ResultSet, convirtiendo a ésta en la fila actual.

El formato general de un ResultSet es una tabla con cabeceras de columna y los valores correspondientes devueltos por la "query". Por ejemplo, si la "query" es *SELECT a, b, c FROM Table1*, el resultado tendrá una forma semejante a:

a	b	c
12345	Argentina	10,5
31245	Brasil	22,7
47899	Perú	56,7

El siguiente fragmento de código es un ejemplo de la ejecución de una sentencia SQL que devolverá una colección de filas, con la columna 1 como un int, la columna 2 como una String y la columna 3 como un array de bytes:

```
Statement stmt = connection.createStatement();
ResultSet r = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (r.next()) {
int i = r.getInt("a");
String s = r.getString("b");
double d = r.getDouble("c");
// imprimimos los valores de la fila actual
System.out.println("Fila = " + i + " " + s + " " + f);
}
```

Filas ResulSet

Un ResultSet mantiene un cursor que apunta a la fila actual de datos. El cursor se mueve una fila hacia abajo cada vez que se llama al método next. Inicialmente se sitúa antes de la primera fila, por lo que hay que llamar al método next para situarlo en la primera fila convirtiéndola en la fila actual. Las filas de ResultSet se recuperan en secuencia desde la fila más alta a la más baja.

Columnas ResultSet

Los métodos getX suministran los medios para recuperar los valores de las columnas de la fila actual. Dentro de cada fila, los valores de las columnas pueden recuperarse en cualquier orden, pero para asegurar la máxima portabilidad, deberían extraerse las columnas de izquierda a derecha y leer los valores de las columnas una única vez.

Puede usarse, o bien el nombre de la columna o el número de columna, para referirse a esta. Por ejemplo: si la columna segunda de un objeto ResultSet rs se denomina "nombre" y almacena valores de cadena, cualquiera de los dos ejemplos siguientes nos devolverá el valor almacenado en la columna.

```
String s = rs.getString("nombre");
String s = rs.getString(2);
```

Nótese que las columnas se numeran de izquierda a derecha comenzando con la columna 1. Además los nombres usados como input en los métodos getX son insensibles a las mayúsculas.

Algunos de los datos que podemos traer con el método get es:

Metodo	Explicación
getInt	Sirve para obtener un numero entero de la base de datos
getLong	Sirve para obtener un numero long de la base de datos
getDouble	Sirve para obtener un numero real de la base de datos
getBoolean	Sirve para obtener un booleano de la base de datos
getString	Sirve para obtener una cadena de la base de datos
getDate	Sirve para obtener una fecha de la base de datos

Optimización de sentencias

Cuando se quiere invocar una determinada sentencia repetidas veces, puede ser conveniente dejar esa sentencia preparada para que pueda ser ejecutada de forma más eficiente. Para hacer esto se utiliza la interfaz PreparedStatement, que podrá obtenerse a partir de la conexión a la BD de la siguiente forma:

```
PreparedStatement ps = con.prepareStatement("SELECT * FROM nombreTabla WHERE campo2> 1200 AND campo2< 1300");
```

Vemos que, a este objeto, a diferencia del objeto Statement visto anteriormente, se le proporciona la sentencia SQL en el momento de su creación, por lo que estará preparado y optimizado para la ejecución de dicha sentencia posteriormente.

Sin embargo, lo más común es que se necesite hacer variaciones sobre la sentencia, ya que normalmente no será necesario ejecutar repetidas veces la misma sentencia exactamente, sino variaciones de ella. Por ello, este objeto nos permite parametrizar la sentencia. Para ello se deben establecer las posiciones de los parámetros con el carácter '?' dentro de la cadena de la sentencia, tal como se muestra a continuación:

```
PreparedStatement ps = con.prepareStatement("UPDATE FROM nombreTabla
                            SET campo1 = 'valor'
                            WHERE campo2 > ? AND campo2 < ?");
```

En este caso se tienen dos parámetros, que representan un rango de valores en el cual se quiere actualizar. Cuando se ejecute esta sentencia, el campo1 de la tabla nombreTabla se establecerá a valor1 desde el límite inferior hasta límite superior indicado en el campo2.

Para dar valor a estos parámetros se utiliza el método setXXX() donde XXX será el tipo de los datos que asignamos al parámetro, indicando el número del parámetro (que empieza desde 1) y el valor que le queremos dar. Por ejemplo, para asignar valores enteros a los parámetros se debe hacer:

```
ps.setInt(1,1200);
ps.setInt(2,1300);
```

Una vez asignados los parámetros, se puede ejecutar la sentencia llamando al método executeUpdate() del objeto PreparedStatement:

```
int n = ps.executeUpdate();
```

Igual que en el caso de los objetos Statement, se puede utilizar cualquier otro de los métodos para la ejecución de sentencias, executeQuery() o execute(), según el tipo de sentencia que se vaya a ejecutar.

PATRON DE DISEÑO DAO

Para la realización de los ejercicios y en los videos, vamos a trabajar JDBC usando el patrón de diseño DAO. Pero primero debemos saber que es un patrón de diseño.

¿QUE ES UN PATRON DE DISEÑO?

Un patrón de diseño es una solución probada que resuelve un tipo específico de problema en el desarrollo de software referente al diseño.

Las ventajas de usar un patrón de diseño son, que permiten tener el código bien organizado, legible y mantenible, además te permite reutilizar código y aumenta la escalabilidad en tu proyecto. En sí proporcionan una terminología estándar y un conjunto de buenas prácticas en cuanto a la solución en problemas de desarrollo de software.

PATRON DE DISEÑO DAO

A la hora de trabajar con JDBC y trabajar con base de dato, una de las grandes problemáticas al momento de acceder a los datos, es que la implementación y formato de la información puede variar según la fuente de los datos. Implementar la lógica de acceso a datos en la capa de lógica de negocio puedes ser un gran problema, pues tendríamos que lidiar con la lógica de negocio en sí, más la implementación para acceder a los datos

Dado lo anterior, el patrón DAO propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta forma, el DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información; por otra parte, la capa de negocio solo se preocupa por lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.

CLASES

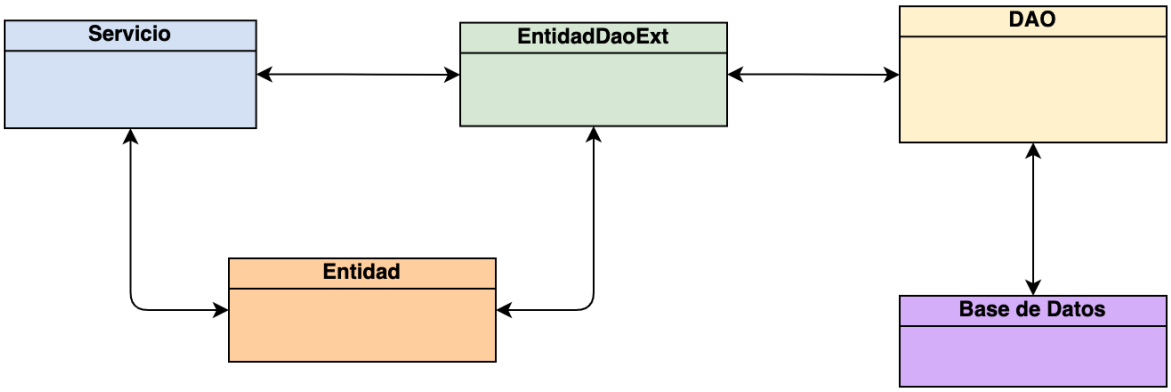
Esto lo vamos a lograr a través de cuatro clases:

Entidad: va a ser la clase que va a representar a la tabla que queremos trabajar de la base de datos. Va tener como atributos las columnas de la tabla de la base de datos.

Servicio o Business Service: va a tener toda la lógica de negocio del proyecto, usualmente se genera una para cada entidad. Es la que se encarga de obtener datos desde la base de datos y enviarla al cliente, o a su vez recibir la clase desde el cliente y enviar los datos al servidor, por lo general tiene todos los métodos CRUD (create, read, update y delete).

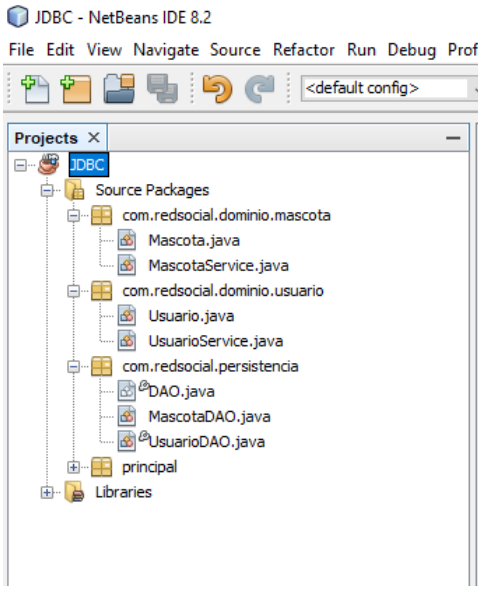
DAO: representa una capa de acceso a datos que oculta la fuente y los detalles técnicos para recuperar los datos. Esta clase va a ser la encargada de comunicarse con la base de datos, de conectarse con la base de datos, enviar las consultas y recuperar la información de la base de datos.

EntidadDaoExt: esta clase va a extender de la clase DAO y se va encargar de generar las sentencias para enviar a la clase DAO, como un insert, select, etc. Y si estuviéramos haciendo un select, sería también, la encargada de recibir la información, que recupera la clase DAO de la base de datos sobre una entidad, para después enviarla al servicio, que será la encargada de imprimir dicha información. Este es un objeto plano que implementa el patrón Data Transfer Object (DTO), el cual sirve para transmitir la información entre el DAO y el Business Service.



PAQUETES

Esto representado en un proyecto tendría las siguientes clases y paquetes:



Nota: estos conceptos van a poder verlos en más profundidad en los videos, donde verán clase por clase y tendrán un ejemplo para descargar y poder verlo ustedes mismos también.

PREGUNTAS DE APRENDIZAJE

1) ¿Que significa el acrónimo JDBC?

   a) Java DataBase Connectivity
   b) Java DataBase Conection
   c) Java DataBag Connectivity
   d) Ninguna de las anteriores

2) La API de JDBC es un conjunto de:

   a) Objetos
   b) Clases e Interfaces
   c) Métodos
   d) Solo Clases

3) El encargado de implementar las clases de JDBC es :

   a) El JDBC Driver
   b) Java
   c) La base de datos
   d) Ninguna de las anteriores

4) El paquete de la API de JDBC es:

   a) java.jdbc.sql.*;
   b) java.mysql.*;
   c) java.sql.jdbc*;
   d) java.sql*;

5) Cual de estas no es una clase de la API JDBC:

   a) Statement
   b) ResultSet
   c) Connection
   d) MetaData

6) ¿Cual es la clase encarga de conectarse con la base de datos?

   a) Statement
   b) ResultSet
   c) Connection
   d) DriverManager

7) ¿Que datos necesita el objeto Connection para conectarse?

   a) url, username y password
   b) username y password
   c) url, username y nombre BD
   d) nombre BD, username, password

8) ¿Cual es la clase encarga de ejecutar las a la base de datos?

a) Statement
b) ResultSet
c) Connection
d) DriverManager

9) ¿Cual es la clase encarga de obtener los datos de la base de datos?

a) Statement
b) ResultSet
c) Connection
d) DriverManager

10) ¿Cual de estas es la excepción de SQL?

a) MysqlException
b) SQLException
c) DataBaseExcepction
d) ConnectionException