MP_0486. Acceso a datos
UF2. Manejo de conectores a bases de datos relacionales

2.3. Acceso a bases de datos con JDBC

Índice

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Conocer las librerías de clases Java relacionadas con el acceso a bases de datos.
- Establecer conexión con una base de datos utilizando el API JDBC.
- Conocer las clases principales de la librería *java.sql*.

Acceso a bases de datos desde java

El API JDBC

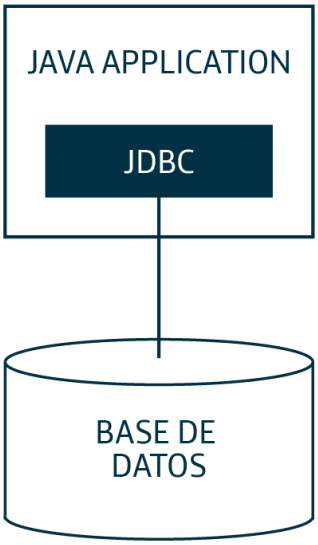
El API JDBC provee un mecanismo sencillo para acceder a bases de datos relacionales desde aplicaciones Java.

Se encuentra disponible en el paquete *java.sql*, que pertenece a la "JRE System Library", por lo que no es necesario cargar ninguna librería externa.

Sin embargo, sí requiere incluir las sentencias ***import*** necesarias para importar las clases que vayamos a utilizar.

La principal característica de JDBC es que **permite independizar nuestro código Java de las peculiaridades de cada DBMS** (Data Base Management System), de modo que nuestro programa Java pueda funcionar exactamente igual accediendo a una base de datos MySQL, una base de datos SQL Server, Oracle, etc.

JDBC se sitúa como una capa dentro de nuestra aplicación Java, capaz de comunicarse con el DBMS.

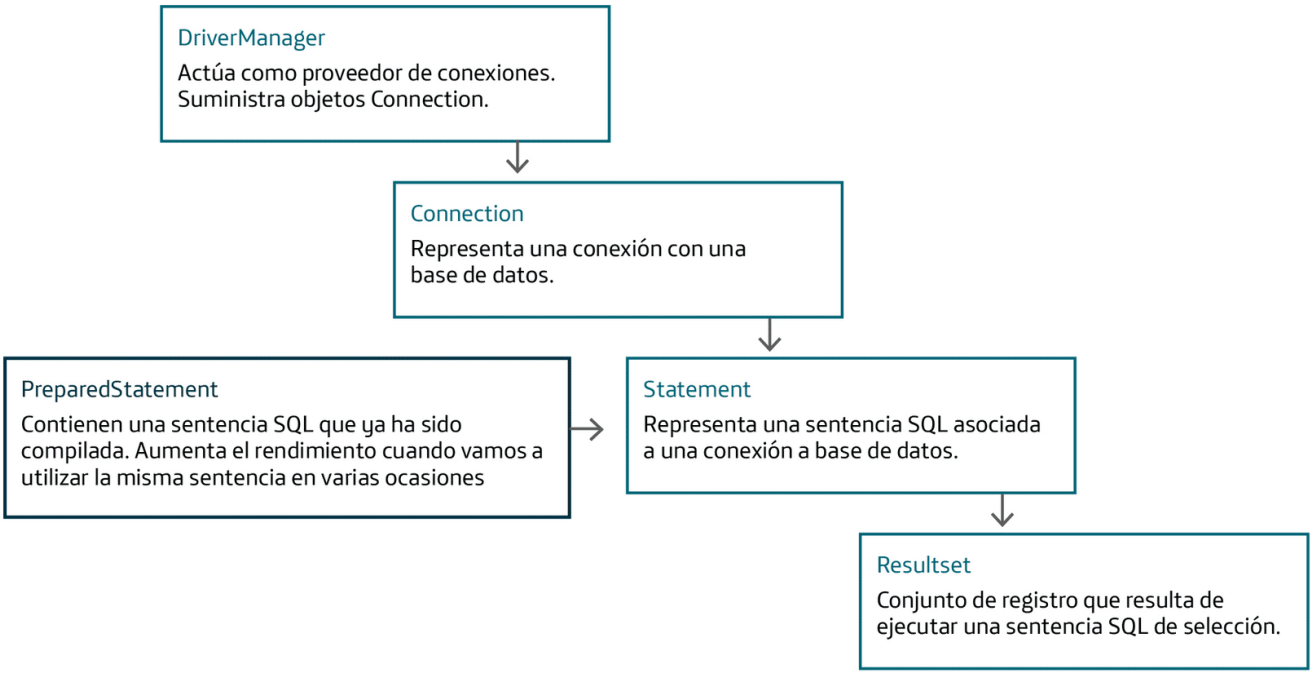


En función del DBMS que soporte la base de datos, JDBC necesitará un ***driver* o conector** especial para Java que habrá que añadir al proyecto.

Para los ejemplos de esta lección nos basaremos en una base de datos MySQL.

Desde nuestro programa Java realizamos llamadas estándar a métodos de las clases del API JDBC, y es el *driver* o conector el que se encarga de traducir estas llamadas a un lenguaje que entienda el DBMS.

Clases más importantes del API JDBC



Esquema de clases del API JDBC.

- ***DriverManager***: como hemos comentado anteriormente, dentro del proyecto es necesario incluir un *driver* o conector por cada DBMS con el que nos queramos comunicar. Pues bien, será la clase *DriverManager* la que se encargue de gestionar estos *drivers* y de proveer al programa de las bases de datos que necesite.
- ***Connection***: representa una conexión con una base de datos concreta, que será previamente suministrada por el *DriverManager*. La conexión con la base de datos se realiza a través de una cadena de conexión, que contendrá los datos de conexión necesarios.
- ***Statement***: permite ejecutar una sentencia SQL, que podrá ser de selección de datos (*SELECT*), de actualización (*INSERT*, *DELETE*, *UPDATE*) o de definición (CREATE, ALTER, DROP).
- ***PreparedStatement***: en ocasiones tenemos que ejecutar muchas sentencias SQL que son iguales, pero con distintos datos o parámetros. Por ejemplo: si tenemos que añadir 200 artículos en una tabla necesitaremos ejecutar 200 sentencias *INSERT* exactamente iguales, donde sólo varían los datos de cada artículo. La ejecución de cada uno de estos *INSERT* supone para el DBMS realizar las siguientes tareas:
    - Comprobar que la sentencia SQL es correcta.
    - Convertir los datos al tipo adecuado para el DBMS.
    - Ejecutar la sentencia SQL.

Realizar estas tareas 200 veces podría suponer un tiempo considerable. Para hacer más eficiente este proceso ***PreparedStatement*** **nos brinda un sistema para precompilar la sentencia SQL y guardarla para ser ejecutada inmediatamente, sin necesidad de analizarla en cada caso.**

PreparedStatement también nos permite añadir seguridad a nuestros programas, ya que podemos evitar "inyecciones SQL". Pulsa aquí para conocer lo que son las inyecciones SQL.

- ***ResultSet***: los objetos *ResultSet* se crean como resultado de la ejecución de una sentencia *SELECT* y contienen el conjunto de registros resultado de su ejecución.

Primer proyecto con acceso a base de datos

Descargar el conector y crear el proyecto

En este apartado aprenderás a crear el proyecto Java que utilizaremos a lo largo de esta lección, descargarás de Internet el *driver* o conector para MySQL e importarás el *driver* dentro de dicho proyecto.

Antes de realizar el acceso desde Java a cualquier base de datos relacional a través de JDBC, es necesario realizar estas tres tareas:

- Descargar de Internet el *driver* o conector.
- Importar el *driver* o conector en el proyecto Java que deba acceder a la base de datos. Este *driver* será distinto en función del DBMS utilizado.
- Cargar en memoria el *driver*, ejecutando el método *Class.forName(...)* desde el código Java.

En este caso, vamos a ir directamente al paso 3 para ver qué ocurre. Crea un proyecto Java nuevo llamado *Unidad2* y, dentro la clase *Principal*, introduce el siguiente código:

```
public class Principal {

	public static void main(String[] args) {

		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
		 System.out.println("No se ha encontrado el driver para MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");

	}

}
```

Class.forName("com.mysql.jdbc.Driver");
Con esta línea estamos cargando en memoria el *driver* de MySQL para Java. *Class.forName(...)* sirve para cargar en memoria un software específico, en este caso el *driver* o conector de MySQL. Pero no hemos descargado el *driver* ni lo hemos importado en el proyecto, por lo que ha saltado la excepción *ClassNotFoundException*, lo que ha provocado que el flujo del programa salte al bloque *catch(...)* y finalice ahí con la sentencia *return*.

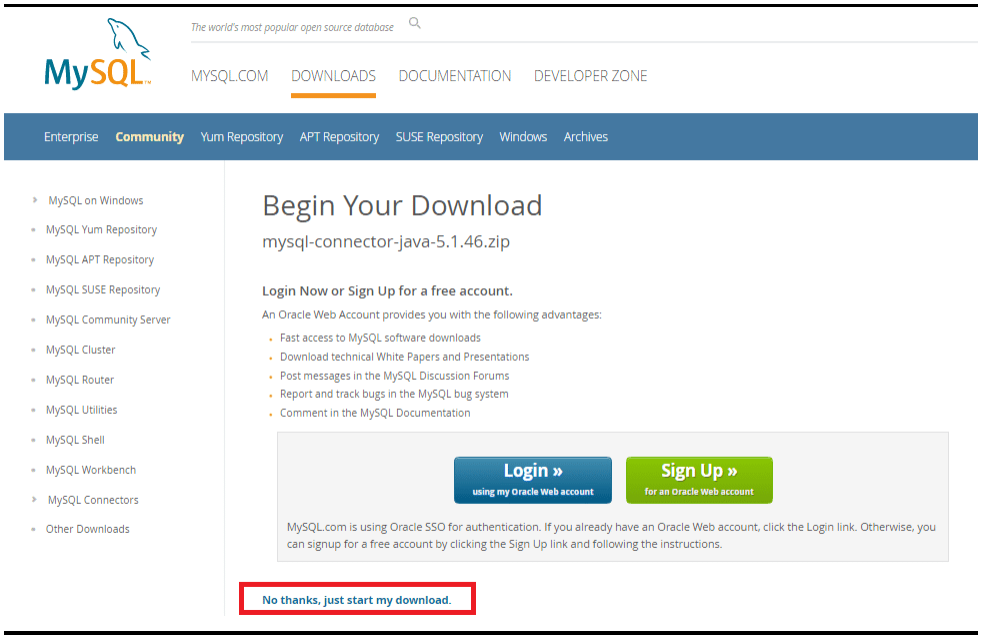
Descargar el *driver* para MySQL

Conector MySQL para Java
Pulsa en el botón de la derecha para descargar el *driver*o conector de MySQL para Java desde la web oficial de MySQL. https://dev.mysql.com/downloads/connector/j/

Localiza el área de descargas, moviéndote hacia abajo con la barra de desplazamiento si es necesario. El área de descargas tendrá un aspecto similar a éste:

Aspecto del área de descargas.

Pulsa el botón "***Download***" para comenzar la descarga del archivo "***mysql-connector-java-5.1.46.zip***". Seguramente no comenzará directamente, sino que entrarás en otra página similar a esta:



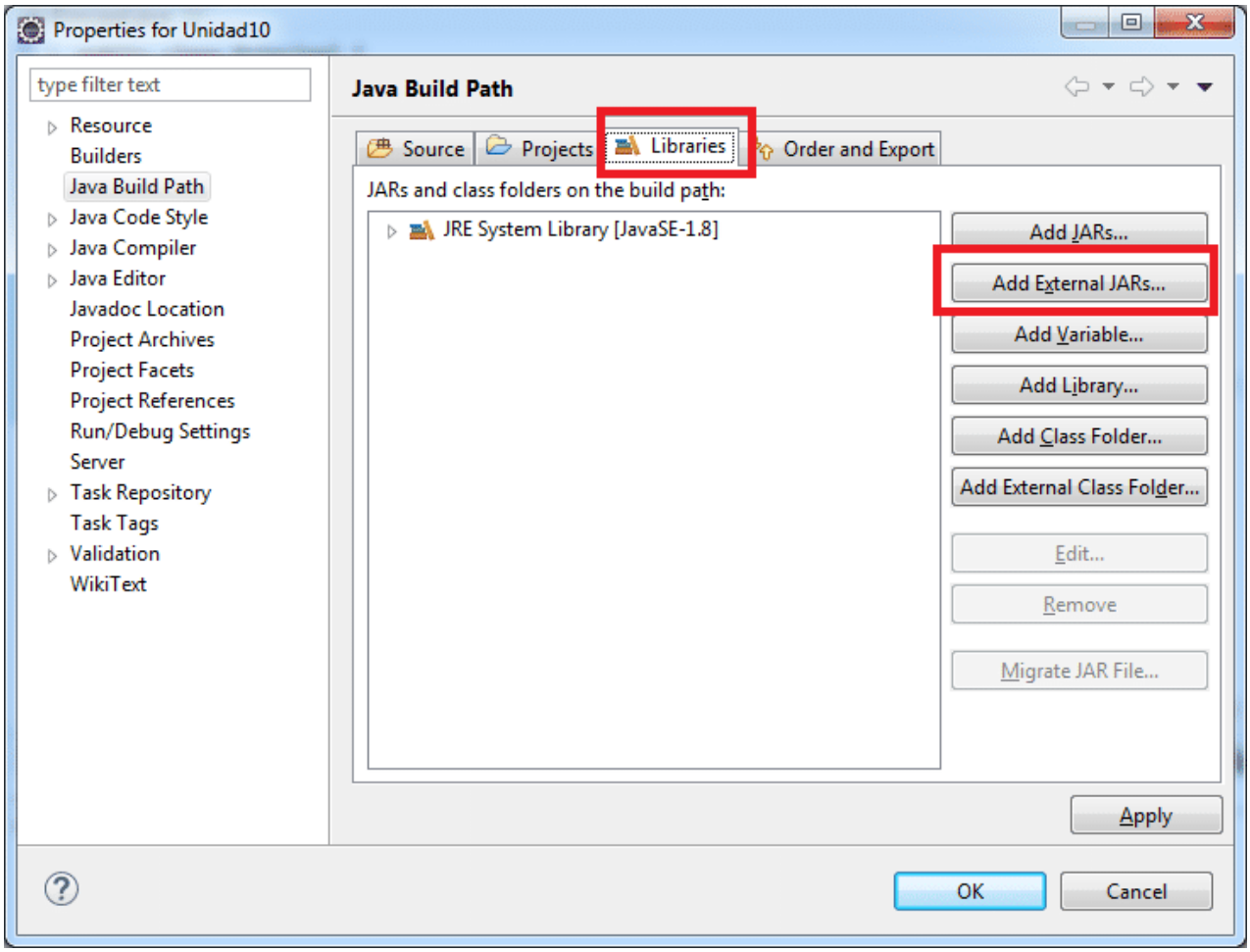
Descarga del conector de MySQL para Java.

Esta página te propone que te registres en la web, pero no es necesario, así que pulsa sobre el enlace "***No thanks, just start my download***". Cuando se haya terminado de descargar el archivo "*mysql-connector-java-5.1.46.zip*", descomprímelo en la ubicación que desees para obtener una carpeta con nombre "***mysql-connector-java-5.1.46***".

Ya has completado el primer paso; la descarga del *driver*. Si trabajas con otro DBMS, el proceso no será muy distinto. Se trata de acceder a la web oficial del DBMS y buscar el enlace que permita descargar el *driver* para Java.

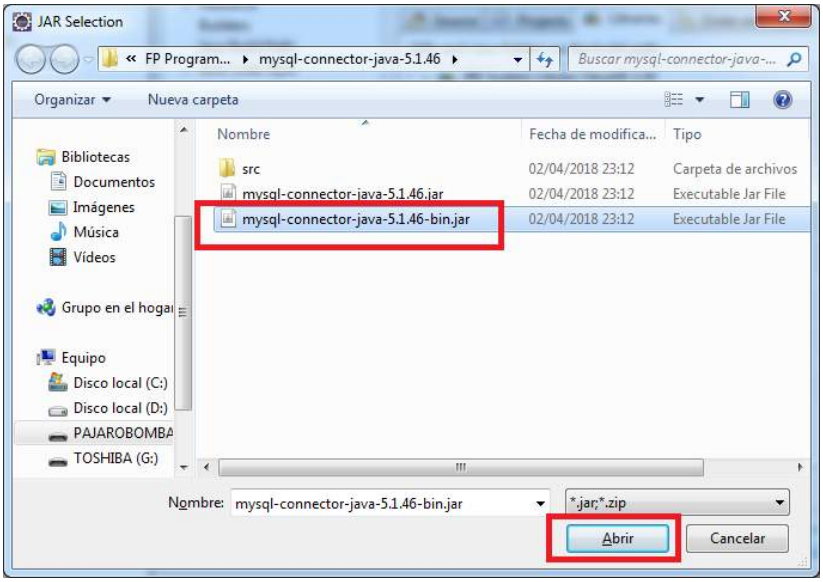
Importar el *driver* en el proyecto Java

Haz **clic derecho** sobre el nombre del proyecto *Unidad2* para obtener el menú contextual y selecciona la opción "***Properties***", que se encuentra al final del todo. Te encontrarás en el cuadro de diálogo "***Properties for Unidad2***".

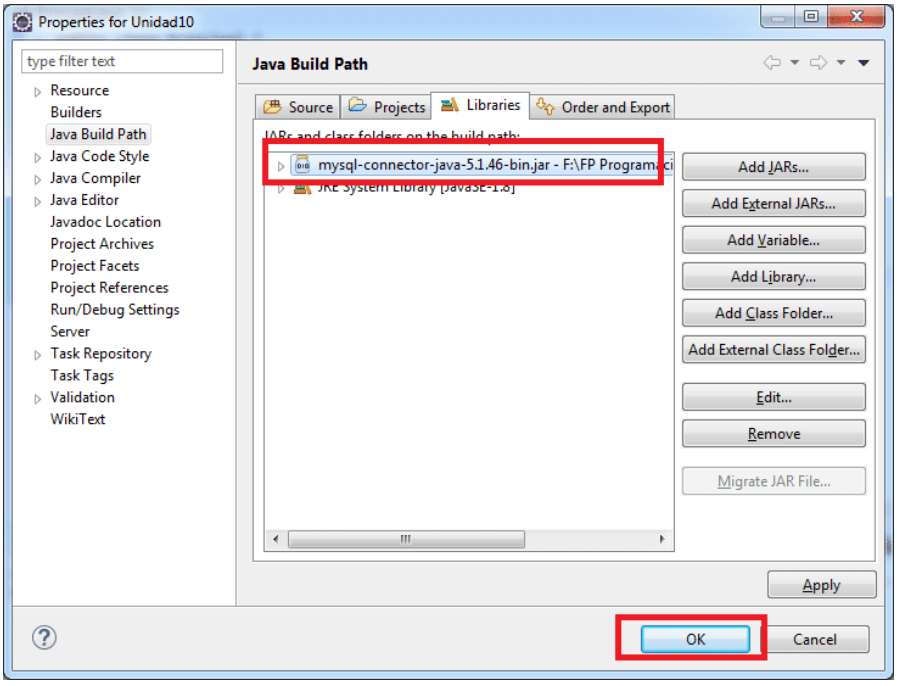


Project properties dialog.

Activa la ficha "***Libraries***" y haz clic en el botón "***Add External JARs...***" para abrir el cuadro de diálogo "***JAR Selection***", donde debes navegar a través del sistema de archivos de tu equipo hasta encontrar la carpeta que resultó de la descompresión del archivo *.zip* con el *driver*.

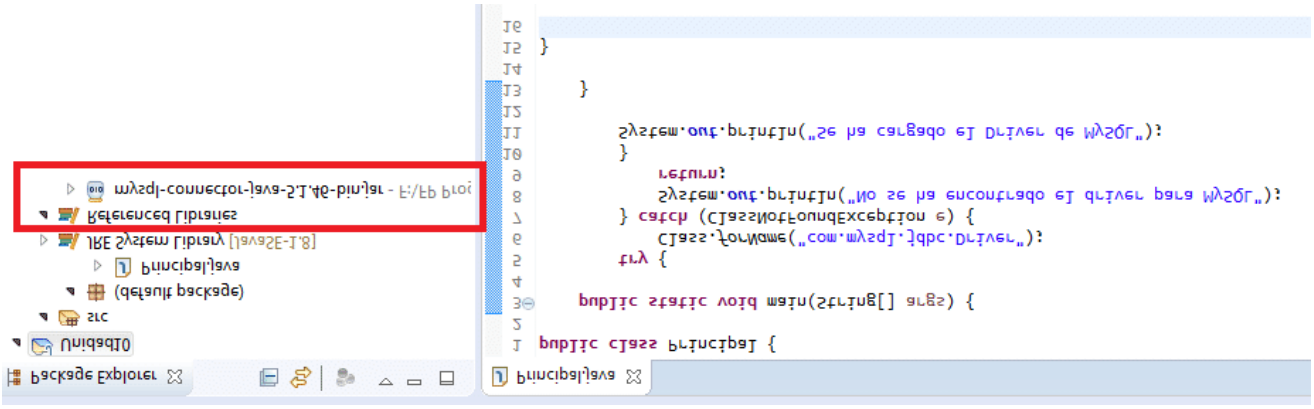


Selecciona el archivo "***mysql-connector-java-5.1.46-bin.jar***" y haz clic sobre el botón "***Abrir***". Vuelves así, de nuevo, al cuadro de diálogo "***Properties for Unidad2***", donde puedes comprobar que se ha incluido el archivo.



Aquí ya sólo tienes que pulsar el botón "***OK***" y el *driver* estará incluido en el proyecto.

Ahora, puedes observar de nuevo tu proyecto y ver que se ha incluido la carpeta "***Referenced Libraries***" con el nuevo *driver*.



Si ejecutas de nuevo el programa, comprobarás que, de momento, no hace gran cosa, sólo muestra el mensaje "*Se ha cargado el Driver de MySQL*". Pero, en realidad, hemos dado un paso muy importante: **nuestro programa ya está preparado para comenzar a trabajar con el API JDBC para comunicarse con MySQL.**

Conectar con la base de datos

En este apartado vamos a establecer conexión con una base de datos MySQL llamada FERRETERIA.

Todos los ejemplos de código Java de esta unidad se conectarán a la base de datos FERRETERIA que creaste en la unidad anterior.

Para lograr conectar a la base de datos *FERRETERIA* utilizaremos las siguientes clases:

- ***DriverManager***: contiene información de todos los *drivers* o conectores cargados en memoria y actúa como proveedor, suministrando conexiones siempre y cuando tenga acceso a los *drivers* necesarios. Recuerda que hemos preparado el camino al *DriverManager* para que encuentre dichos *drivers* en el apartado anterior.
- ***Connection***: representa una conexión con una base de datos determinada y es suministrada por el *DriverManager*.

Para establecer conexión con una base de datos se utiliza el método estático ***getConnection()*** de la clase *DriverManager*, que retorna un objeto *Connection*. El método *getConnection()* tiene el siguiente formato:

```
Connection nombreObj = DriverManager.getConnection(String cadenaConexion, String
usuario, String contraseña);
```

Formato del método *getConnection*.

El primer argumento es la **cadena de conexión**, que contendrá información sobre la base de datos a la que deseamos acceder. Los argumentos segundo y tercero contienen el nombre de usuario y contraseña del usuario autorizado a utilizar la base de datos.

La cadena de conexión

Una **cadena de conexión** (***connection string***) es un *String* que contiene información de acceso a una base de datos. El formato utilizado es diferente en función del proveedor de bases de datos. **JDBC** nos permite utilizar el mismo código para acceder a un gran número de formatos de bases de datos, cambiando solamente la cadena de conexión.
Puesto que los elementos y el formato de la cadena de conexión varían en función del proveedor de bases de datos, es difícil explicarlo de forma específica. En términos generales, **toda cadena de conexión se divide en las siguientes áreas:**

Proveedor:
Hace referencia al gestor de base de datos utilizado (*jdbc:sqlserver*, *jdbc:derby*, *jdbc:mysql*, *etc.*).

Dirección IP o nombre de servidor:
Se refiere al servidor donde está alojada la base de datos. Se utiliza en SGBD como SQL Server, Oracle o MySQL, que trabajan con tecnología cliente/servidor. Puede ser una DNS o una dirección IP.

Nombre de la instancia:
En SQL Server o MySQL un servidor puede contener varias instancias, cada instancia con sus propias bases de datos.

Número de puerto:
Necesario para establecer la conexión con el servidor. En MySQL es habitual que sea el 3306.

Nombre de la base de datos:
Nombre de la base a la que queremos acceder en nuestro programa.

Y ahora, pongámonos manos a la obra.

Completa el código de la clase *Principal* para establecer la conexión con la base de datos:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Principal {

	public static void main(String[] args) {
```

```
		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");

		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";
		String user = "root";
		String pass = "amelia"; // sustituye por la contraseña que especificaste
durante la instalación de MySQL.
		Connection con;
		try {
			con = DriverManager.getConnection(cadenaConexion, user, pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la conexión con la
BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha establecido la conexión con la Base de datos");

		// Paso 3: Interactuar con la BD (todavía pendiente)


		// Paso 4: Cerrar la conexión
		try {
			con.close();
		} catch (SQLException e) {
			System.out.println("No se ha podido cerrar la conexión con la BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha cerrado la base de datos");

	}

}
```

Establecemos conexión con la base de datos ejecutando la sentencia:
con = DriverManager.getConnection(cadenaConexion, user, pass);

Esta sentencia puede provocar una excepción de tipo ***SQLException*** si no se puede establecer la conexión por múltiples motivos (el usuario no está autorizado, la base de datos no existe, el servidor de BD está realizando tareas de mantenimiento, etc.).

Recuerda sustituir en la sentencia que sigue el valor "amelia" por el password que especificaste durante la instalación de MySQL.
String pass = "amelia";

Hemos utilizado la siguiente cadena de conexión:

Cadena de conexión para acceso a la base de datos MySQL FERRETERIA.

En este caso no hemos incluido nombre de instancia, ya que accedemos a un servidor MySQL que tiene una única instancia y no es necesario.

Como has podido comprobar con este ejemplo, cualquier programa típico que conecta a una base de datos se compone de cuatro partes:

- Cargar el *driver* en memoria.
- Establecer conexión con la base de datos, solicitando al *DriverManager* un objeto *Connection* y suministrando una cadena de conexión.
- Interactuar con la base de datos. Esta es la parte del programa que todavía tenemos pendiente.
- Cerrar la conexión con la base de datos.

Obtener un listado de clientes

En este apartado trabajaremos con las clases *Statement* y *ResultSet*para realizar un listado de clientes.

Vamos a completar el código de nuestra clase *Principal* y después lo analizaremos.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Principal {

	public static void main(String[] args) {

		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");
```

```
		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";
		String user = "root";
		String pass = "amelia";
		Connection con;
		try {
			con = DriverManager.getConnection(cadenaConexion, user, pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la conexión con la
BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha establecido la conexión con la Base de datos");

		// Paso 3: Interactuar con la BD
		try {
			Statement sentencia = con.createStatement();
			ResultSet rs = sentencia.executeQuery("SELECT * FROM CLIENTE");
			while (rs.next()) {
				System.out.print(rs.getString("NIF"));
				System.out.print(" - ");
				System.out.print(rs.getString("NOMBRE"));
				System.out.print(" - ");
				System.out.print(rs.getString("TLF"));
				System.out.println(); // Retorno de carro
			}
		} catch (SQLException e) {
			System.out.println("Error al realizar el listado de productos");
			System.out.println(e.getMessage());
		}

		// Paso 4: Cerrar la conexión
		try {
			con.close();
		} catch (SQLException e) {
			System.out.println("No se ha podido cerrar la conexión con la BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha cerrado la base de datos");

	}

}
```

Ahora, si ejecutas el programa obtendrás en pantalla un **listado de clientes**.

La parte del código que nos interesa analizar en este momento es la siguiente:

```
// Paso 3: Interactuar con la BD
		try {
			Statement sentencia = con.createStatement();
			ResultSet rs = sentencia.executeQuery("SELECT * FROM CLIENTE");
			while (rs.next()) {
				System.out.print(rs.getString("NIF"));
				System.out.print(" - ");
				System.out.print(rs.getString("NOMBRE"));
```

```
                    System.out.print(" - ");
                    System.out.print(rs.getString("TLF"));
                    System.out.println(); // Retorno de carro
                }
            } catch (SQLException e) {
                System.out.println("Error al realizar el listado de productos");
                System.out.println(e.getMessage());
            }
```

Por último, analicemos las **partes más importantes del programa:**

Statement sentencia = con.createStatement();

Será labor del objeto *Connection* (para nosotros la variable *con*) proveernos de un objeto de la clase *Statement* que nos servirá para ejecutar sentencias SQL. Llamaremos a nuestro objeto *Statement sentencia*.

ResultSet rs = sentencia.executeQuery("SELECT * FROM CLIENTE");

Los objetos *Statement* disponen del método *executeQuery()* para ejecutar sentencias de tipo *SELECT*, y retornan un objeto *ResultSet* con el conjunto de registros resultante. Dentro del programa, *rs* es nuestro objeto *ResultSet*. Para ejecutar otro tipo de sentencias, como *INSERT*, *UPDATE*, *DELETE*, etc., se utiliza el método *executeUpdate()*.

while (rs.next()) { .... }

El objeto *ResultSet* actúa como un cursor que permite desplazarse a través del conjunto de registros hacia delante, hacia atrás, al primero y al último. El método *next()* sitúa el puntero en el siguiente registro y retorna *true* si no es final de fichero, con lo que podemos utilizar su resultado para realizar un recorrido secuencial mientras siga retornando *true*.

System.out.print(rs.getString("NIF"));

El objeto *ResultSet* dispone de varios métodos para recuperar una columna y campo del registro activo: *getString* (para recuperar un campo de tipo cadena), *getInt* (para recuperar un campo de tipo Integer), *getFloat* (para recuperar un campo de tipo float), etc.

En función del tipo de datos con que se definió el campo en el DBMS, habrá que elegir un método u otro. Recuerda los tipos de datos de MySQL que estudiamos en la unidad anterior.

Estos métodos admiten como argumento una cadena con el nombre del campo, o bien un número entero, que representa el número de orden de la columna. La sentencia podría haberse escrito así: "*System.out.print(rs.getString(1));*" haciendo el 1 referencia a la primera columna, que es el *NIF*.

Añadir un nuevo cliente

Vamos a añadir un nuevo cliente a la tabla CLIENTE.

Puedes sustituir los datos que aparecen en el ejemplo por tus propios datos u otros inventados, y luego volver a ejecutar el ejemplo anterior del listado para comprobar que realmente se ha añadido el registro.

Comienza por añadir a tu proyecto la clase *PrincipalNuevo* con el código que podrás copiar y pegar desde aquí; luego, lo analizaremos paso a paso.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class PrincipalNuevo {

	public static void main(String[] args) {

		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");

		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";
		String user = "root";
		String pass = "amelia";
		Connection con;
		try {
			con = DriverManager.getConnection(cadenaConexion, user, pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la conexión con la
BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha establecido la conexión con la Base de datos");

		// Paso 3: Interactuar con la BD
		try {
			String nif = "55667788A";
			String nombre = "DELGADO PEREZ CARLOS";
			String domicilio = "C/ ALENZA, 7";
			String tlf = "616667766";
			String ciudad = "MADRID";
			String sql = "INSERT INTO CLIENTE " +
				"(NIF, NOMBRE, DOMICILIO, TLF, CIUDAD) " +
				"VALUES ('" + nif + "', '" + nombre +
				"', '" + domicilio + "', '" + tlf +  "', '" +
				ciudad + "');";
```

```
                System.out.println("Se va a ejecutar la siguiente sentencia SQL:");
                System.out.println(sql);
                Statement sentencia;
                sentencia = con.createStatement();
                int afectados = sentencia.executeUpdate(sql);
                System.out.println("Sentencia SQL ejecutada con éxito");
                System.out.println("Registros afectados: "+afectados);
            } catch (SQLException e) {
                System.out.println("Error al añadir nuevo cliente");
                System.out.println(e.getMessage());
            }


            // Paso 4: Cerrar la conexión
            try {
                con.close();
            } catch (SQLException e) {
                System.out.println("No se ha podido cerrar la conexión con la BD");
                System.out.println(e.getMessage());
                return;
            }
            System.out.println("Se ha cerrado la base de datos");

        }

}
```

El objetivo de este programa es **enviar una orden al DBMS de MySQL para que ejecute la siguiente sentencia SQL**:

```
INSERT INTO CLIENTE (NIF, NOMBRE, DOMICILIO, TLF, CIUDAD) VALUES ('55667788A',
'DELGADO PEREZ CARLOS', 'C/ ALENZA, 7', '616667766', 'MADRID';
```

¡Ojo! Los valores de tipo texto en SQL deben ir encerrados entre comillas simples.

Los datos del nuevo cliente se encuentran en varias variables tipo *String*, y hemos tenido que montar la sentencia SQL concatenando los valores de dichas variables dentro de la sentencia y respetando la sintaxis requerida.

```
                String nif = "55667788A";
                String nombre = "DELGADO PEREZ CARLOS";
                String domicilio = "C/ ALENZA, 7";
                String tlf = "616667766";
                String ciudad = "MADRID";
                String sql = "INSERT INTO CLIENTE " +
                      "(NIF, NOMBRE, DOMICILIO, TLF, CIUDAD) " +
                      "VALUES ('" + nif + "', '" + nombre +
                      "', '" + domicilio + "', '" + tlf +  "', '" +
                      ciudad + "');";
```

Una vez montada la sentencia SQL, la hemos ejecutado de nuevo con ayuda de un objeto *Statement*.

```
sentencia = con.createStatement();
                int afectados = sentencia.executeUpdate(sql);
```

Pero esta vez, en lugar de utilizar el método *executeQuery(...)*, hemos utilizado el método ***executeUpdate(...)***, que **sirve para ejecutar una sentencia de actualización tipo *INSERT*, *UPDATE* o *DELETE*** y retorna el número de filas de la tabla que han sido afectadas, es decir, insertadas, borradas o modificadas.

Hemos añadido un solo cliente, pero recuerda que si tuviéramos que añadir 200 clientes mediante este sistema, tendríamos que ejecutar 200 sentencias iguales, donde sólo cambiarían los datos personales de las 200 personas. Por cada persona añadida, el DBMS tendría que comprobar la sintaxis de la sentencia, tarea que podría tener un coste considerable en tiempo de ejecución.

En el siguiente apartado realizaremos el mismo programa, pero usando la clase *PreparedStatement*, que nos permitirá mejorar considerablemente el rendimiento cuando ejecutemos la misma sentencia en numerosas ocasiones.

Usar PreparedStatement

PreparedStatement nos brinda un sistema para precompilar la sentencia SQL y guardarla para ser ejecutada inmediatamente, sin necesidad de analizarla en cada caso.

A continuación, expondremos el mismo programa del apartado anterior, pero usando la clase *PreparedStatement*:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class PrincipalNuevo {

	public static void main(String[] args) {

		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");
```

```
		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";
		String user = "root";
		String pass = "amelia";
		Connection con;
		try {
			con = DriverManager.getConnection(cadenaConexion, user, pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la conexión con la
BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha establecido la conexión con la Base de datos");

		// Paso 3: Interactuar con la BD
		try {

			String sql = "INSERT INTO CLIENTE (NIF, NOMBRE, DOMICILIO, TLF,
CIUDAD) " +
				"VALUES (?, ?, ?, ?, ?)";

			String nif = "55667788A";
			String nombre = "DELGADO PEREZ CARLOS";
			String domicilio = "C/ ALENZA, 7";
			String tlf = "616667766";
			String ciudad = "MADRID";
			System.out.println("Se va a ejecutar la siguiente sentencia SQL:");
			System.out.println(sql);
			PreparedStatement sentencia;
			sentencia = con.prepareStatement(sql);
			sentencia.setString(1, nif);
			sentencia.setString(2, nombre);
			sentencia.setString(3, domicilio);
			sentencia.setString(4, tlf);
			sentencia.setString(5, ciudad);
			int afectados = sentencia.executeUpdate();
			System.out.println("Sentencia SQL ejecutada con éxito");
			System.out.println("Registros afectados: "+afectados);
		} catch (SQLException e) {
			System.out.println("Error al añadir nuevo cliente");
			System.out.println(e.getMessage());
		}


		// Paso 4: Cerrar la conexión
		try {
			con.close();
		} catch (SQLException e) {
			System.out.println("No se ha podido cerrar la conexión con la BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha cerrado la base de datos");

	}

}
```

Ahora vamos a analizar el código:

En primer lugar, *PreparedStatement* nos permite crear una sentencia SQL con parámetros que posteriormente podemos reemplazar:

```
String sql = "INSERT INTO CLIENTE (NIF, NOMBRE, DOMICILIO, TLF, CIUDAD) " +
"VALUES (?, ?, ?, ?, ?)";
```

Cada interrogación es un parámetro que habrá que sustituir por los datos de un cliente. De esta forma nos sirve la misma cadena de texto para todas las sentencias del mismo tipo que vayamos a ejecutar.

```
PreparedStatement sentencia;
sentencia = con.prepareStatement(sql);
```

Luego, hemos creado un objeto de tipo *PreparedStatament* asociado a la conexión abierta y lo hemos pasado como argumento al constructor la cadena SQL con los parámetros pendientes de reemplazar.

```
sentencia.setString(1, nif);
sentencia.setString(2, nombre);
sentencia.setString(3, domicilio);
sentencia.setString(4, tlf);
sentencia.setString(5, ciudad);
```

Por cada una de las interrogaciones (parámetros) dentro de la cadena SQL, tenemos que ejecutar un método *set...* con el siguiente formato:

obj.setxxx(ordenParametro, valorParametro);

Donde *xxx* hace referencia al tipo de datos (*setInt(...)*, *setString(...)*, *setFloat(...)*, *setDouble(...)*, etc.).

```
int afectados = sentencia.executeUpdate();
```

El último paso es ejecutar la sentencia con los métodos *executeQuery()* o *excuteUpdate()*.

Borrar un cliente

En el siguiente apartado ejecutaremos una sentencia SQL de eliminación, es decir, una sentencia *DELETE*.

Eliminaremos de la tabla *CLIENTE* el registro previamente añadido y utilizaremos de nuevo la clase *PreparedStatement*.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class PrincipalBorrar {

	public static void main(String[] args) {

		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para MySQL");
			return;
		}
		System.out.println("Se ha cargado el Driver de MySQL");

		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";
		String user = "root";
		String pass = "amelia";
		Connection con;
		try {
			con = DriverManager.getConnection(cadenaConexion, user, pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la conexión con la
BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha establecido la conexión con la Base de datos");

		// Paso 3: Interactuar con la BD
		try {

			String sql = "DELETE FROM CLIENTE WHERE NIF=?";

			String nif = "55667788A";
			System.out.println("Se va a ejecutar la siguiente sentencia SQL:");
			System.out.println(sql);
			PreparedStatement sentencia;
			sentencia = con.prepareStatement(sql);
			sentencia.setString(1, nif);
			int afectados = sentencia.executeUpdate();
			System.out.println("Sentencia SQL ejecutada con éxito");
			System.out.println("Registros afectados: "+afectados);
		} catch (SQLException e) {
			System.out.println("Error al borrar el cliente");
			System.out.println(e.getMessage());
		}


		// Paso 4: Cerrar la conexión
		try {
			con.close();
		} catch (SQLException e) {
```

```
                    System.out.println("No se ha podido cerrar la conexión con la BD");
                    System.out.println(e.getMessage());
                    return;
             }
             System.out.println("Se ha cerrado la base de datos");

      }

}
```

Vamos a analizar las partes más importante del código:

- Hemos creado una cadena de texto SQL con un parámetro que recogerá el NIF del cliente que queremos borrar:
  String sql = "DELETE FROM CLIENTE WHERE NIF=?";
- Hemos creado un objeto *PreparedStatement* asociado con la conexión abierta, pasando como argumento al constructor la sentencia SQL con el parámetro:
  String nif = "55667788A";
  PreparedStatement sentencia;
  sentencia = con.prepareStatement(sql);
- Hemos asignado valor al parámetro ejecutando la sentencia:
  sentencia.setString(1, nif);
- Hemos ejecutado la sentencia SQL con la siguiente línea de código:
  int afectados = sentencia.executeUpdate();

Programar la gestión de clientes

Vamos a ver un ejemplo completo de gestión de la tabla CLIENTE a partir de un menú que sirve como interfaz para el usuario.

Puedes crear un nuevo proyecto Java llamado ***GestionClientes***, donde añadirás la siguiente clase:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class Principal {
      private static Scanner lector;

      public static void main(String[] args) {

            Connection con = abrirConexionBD();
            if (con == null)
                  return;
```

```
		lector = new Scanner(System.in);
		int opc;

		do {
			mostrarMenu();
			opc = lector.nextInt();
			lector.nextLine(); // Recogemos el retorno de carro.
			System.out.println();
			switch (opc) {
			case 1:
				listarClientes(con);
				break;
			case 2:
				nuevoCliente(con);
				break;
			case 3:
				borrarCliente(con);
				break;
			case 4:
				modificarCliente(con);
				break;
			case 5:
				buscarCliente(con);
				break;
			case 6:
				System.out.println("Hasta pronto");
				break;
			default:
				System.out.println("Opción incorrecta");
			}
		} while (opc != 6);

		cerrarConexion(con); // Pasamos como argumento la conexión a cerrar.
		lector.close();
	}

	private static Connection abrirConexionBD() {
		Connection con;
		// Paso 1: Cargar el driver
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.out.println("No se ha encontrado el driver para MySQL");
			return null;
		}
		System.out.println("Se ha cargado el Driver de MySQL");

		// Paso 2: Establecer conexión con la base de datos
		String cadenaConexion = "jdbc:mysql://localhost:3306/FERRETERIA";
		String user = "root";
		String pass = "amelia";
		try {
			con = DriverManager.getConnection(cadenaConexion, user, pass);
		} catch (SQLException e) {
			System.out.println("No se ha podido establecer la conexión con la
BD");
			System.out.println(e.getMessage());
			return null;
		}
		System.out.println("Se ha establecido la conexión con la Base de datos");
		return con; // Devolvemos la conexión abierta.
	}
```

```
	private static void cerrarConexion(Connection con) {
		try {
			con.close();
		} catch (SQLException e) {
			System.out.println("No se ha podido cerrar la conexión con la BD");
			System.out.println(e.getMessage());
			return;
		}
		System.out.println("Se ha cerrado la base de datos");
	}

	public static void mostrarMenu() {
		System.out.println();
		System.out.println("GESTION DE CLIENTES");
		System.out.println("--------------------------------------");
		System.out.println("1. Listado de clientes");
		System.out.println("2. Añadir nuevo cliente");
		System.out.println("3. Borrar cliente");
		System.out.println("4. Modificar dirección y teléfono");
		System.out.println("5. Buscar cliente");
		System.out.println("6. Terminar programa");
		System.out.println("--------------------------------------");
		System.out.println("¿Qué opción eliges?");
	}

	private static void listarClientes(Connection con) {
		try {
			Statement sentencia = con.createStatement();
			ResultSet rs = sentencia.executeQuery("SELECT * FROM CLIENTE");
			while (rs.next()) {
				System.out.print(rs.getString(1));
				System.out.print(" - ");
				System.out.print(rs.getString("NOMBRE"));
				System.out.print(" - ");
				System.out.print(rs.getString("DOMICILIO"));
				System.out.print(" - ");
				System.out.print(rs.getString("TLF"));
				System.out.println(); // Retorno de carro
			}
		} catch (SQLException e) {
			System.out.println("Error al realizar el listado de productos");
			System.out.println(e.getMessage());
		}
	}

	private static void nuevoCliente(Connection con) {
		try {
			String sql = "INSERT INTO CLIENTE (NIF, NOMBRE, DOMICILIO, TLF,
CIUDAD) " + "VALUES (?, ?, ?, ?, ?)";
			System.out.println("Introduce NIF del nuevo cliente: ");
			String nif = lector.nextLine();
			System.out.println("Introduce nombre: ");
			String nombre = lector.nextLine();
			System.out.println("Introduce dirección: ");
			String domicilio = lector.nextLine();
			System.out.println("Introduce teléfono: ");
			String tlf = lector.nextLine();
			System.out.println("Introduce ciudad: ");
			String ciudad = lector.nextLine();
			System.out.println("Se va a ejecutar la siguiente sentencia SQL:");
			System.out.println(sql);
```

```
                PreparedStatement sentencia;
                sentencia = con.prepareStatement(sql);
                sentencia.setString(1, nif);
                sentencia.setString(2, nombre);
                sentencia.setString(3, domicilio);
                sentencia.setString(4, tlf);
                sentencia.setString(5, ciudad);
                int afectados = sentencia.executeUpdate();
                System.out.println("Sentencia SQL ejecutada con éxito");
                System.out.println("Registros afectados: " + afectados);
        } catch (SQLException e) {
                System.out.println("Error al añadir nuevo cliente");
                System.out.println(e.getMessage());
        }
}

private static void borrarCliente(Connection con) {
        String sql = "DELETE FROM CLIENTE WHERE NIF=?";
        System.out.println("Escribe NIF del cliente a borrar: ");
        String nif = lector.nextLine();
        PreparedStatement sentencia;
        int afectados;
        try {
                sentencia = con.prepareStatement(sql);
                sentencia.setString(1, nif);
                afectados = sentencia.executeUpdate();
        } catch (SQLException e) {
                System.out.println("No se ha podido borrar el cliente");
                System.out.println(e.getMessage());
                return;
        }
        System.out.println("Sentencia SQL ejecutada con éxito");
        System.out.println("Registros afectados: "+afectados);
}

private static void modificarCliente(Connection con) {
        String sql = "UPDATE CLIENTE SET DOMICILIO=?, TLF=? WHERE NIF=?;";
        System.out.println("Escribe NIF del cliente a modificar: ");
        String nif = lector.nextLine();
        System.out.println("Escribe nueva dirección: ");
        String domicilio = lector.nextLine();
        System.out.println("Escribe nuevo teléfono: ");
        String tlf = lector.nextLine();
        PreparedStatement sentencia;
        int afectados;

        try {
                sentencia = con.prepareStatement(sql);
                sentencia.setString(1, domicilio);
                sentencia.setString(2,  tlf);
                sentencia.setString(3,  nif);
                afectados = sentencia.executeUpdate();
        } catch (SQLException e) {
                System.out.println("No se ha podido modificar el cliente");
                System.out.println(e.getMessage());
                return;
        }
        System.out.println("Sentencia SQL ejecutada con éxito");
        System.out.println("Registros afectados: "+afectados);
}

private static void buscarCliente(Connection con) {
```

```
		try {
			Statement sentencia = con.createStatement();
			System.out.println("Escribe nombre del cliente buscado: ");
			String nombre = lector.nextLine();
			ResultSet rs = sentencia.executeQuery("SELECT * FROM CLIENTE WHERE
NOMBRE LIKE '%" + nombre + "%'");
			while (rs.next()) {
				System.out.print(rs.getString("NIF"));
				System.out.print(" - ");
				System.out.print(rs.getString("NOMBRE"));
				System.out.print(" - ");
				System.out.print(rs.getString("DOMICILIO"));
				System.out.print(" - ");
				System.out.print(rs.getString("TLF"));
				System.out.println(); // Retorno de carro
			}
		} catch (SQLException e) {
			System.out.println("Error al realizar el listado de productos");
			System.out.println(e.getMessage());
		}
	}
}
```

No olvides que, para que funcione, debes importar el *driver* de MySQL dentro del proyecto.

El programa realiza las siguientes tareas:

1. En primer lugar, abre la base de datos llamando al método estático ***abrirConexionBD()***, que devuelve la conexión y deja la base de datos abierta.
2. Una vez abierta la base de datos, el programa permite al usuario realizar las operaciones deseadas, hasta que decida terminar pulsando la opción 6 en el siguiente **menú,** que sirve de interfaz:

```
GESTION DE CLIENTES
--------------------------------------
1. Listado de clientes
2. Añadir nuevo cliente
3. Borrar cliente
4. Modificar dirección y teléfono
5. Buscar cliente
6. Terminar programa
--------------------------------------
¿Qué opción eliges?
```

Según las opciones que vaya seleccionando el cliente, el programa llama a diversos métodos que realizan cada tarea.

3. Por último, se cierra la base de datos llamando al método ***cerrarConexion(con)***, que recibe como argumento el objeto *Connection* a cerrar.

Como ves, el usuario puede realizar varias operaciones con la base de datos, pero **la apertura y cierre de la conexión se realizan una única vez**.

Despedida

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

1. El **API JDBC** provee un mecanismo sencillo para acceder a bases de datos relacionales desde aplicaciones Java. Se encuentra disponible en el paquete ***java.sql***, que pertenece a la "JRE System Library", por lo que no es necesario cargar ninguna librería externa. JDBC se sitúa como una capa dentro de nuestra aplicación Java capaz de comunicarse con el DBMS.
2. La clase ***DriverManager*** se encarga de gestionar los *drivers* de los distintos proveedores de BD y de proveer al programa del objeto *Connection* para establecer la conexión con la BD.
3. La clase ***Connection*** representa una conexión con una base de datos.
4. La clase ***Statement*** permite ejecutar sentencias SQL sobre una conexión a una BD abierta.
5. ***PreparedStatement*** nos brinda un sistema para precompilar la sentencia SQL y guardarla para ser ejecutada inmediatamente, sin necesidad de analizarla en cada caso.
6. Los objetos de tipo ***ResultSet*** se crean como resultado de la ejecución de una sentencia *SELECT* y contienen el conjunto de registros resultado de su ejecución.