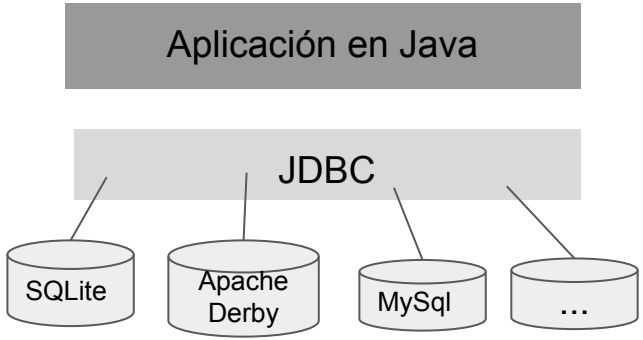
Protocolos de acceso a bases de datos

Introducción

- En tecnologías de base de datos podemos encontrarnos con dos normas de conexión a una base de datos:
  - **ODBC**: Open Database Connectivity. Define una API que pueden usar las aplicaciones para abrir una conexión con una base de datos, enviar consultas, actualizaciones y obtener resultados. Las aplicaciones pueden usar esta API siempre y cuando el servidor de base de datos sea compatible con ODBC
  - **JDBC.** Java Database Connectivity. Define una API que pueden usar los programas Java para conectarse a los servidores de bases de datos relacionales
  - **OLE-DB**. Object Linking and Embedding for Databases. De Microsoft. Es una API de C++ con objetivos parecidos a los de ODBC pero para orígenes de datos que no son bases de datos. En ODBC los comandos siempre están en SQL, en OLE-DB pueden estar en cualquier lenguaje soportado por el origen de datos.

Acceso a datos mediante JDBC

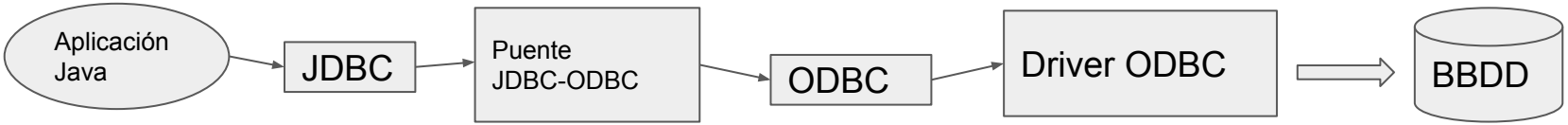
- JDBC proporciona una librería estándar para acceder a fuentes de datos principalmente orientados a bases de datos relacionales que usan SQL.
- No solo provee una interfaz sino que también define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones Java el acceso a los datos.
- JDBC dispone de una interfaz distinta para cada base de datos, es lo que llamamos **driver** (controlador o conector). Esto permite que las llamadas a los métodos Java de las clases JDBC se correspondan con el API de la base de datos.



- JDBC consta de un conjunto de clases e interfaces que nos permiten escribir aplicaciones Java para gestionar las siguientes tareas con una bbdd relacional:
  - Conectarse a la base de datos
  - Enviar consultas e instrucciones DML a la bbdd
  - Recuperar y procesar los resultados recibidos

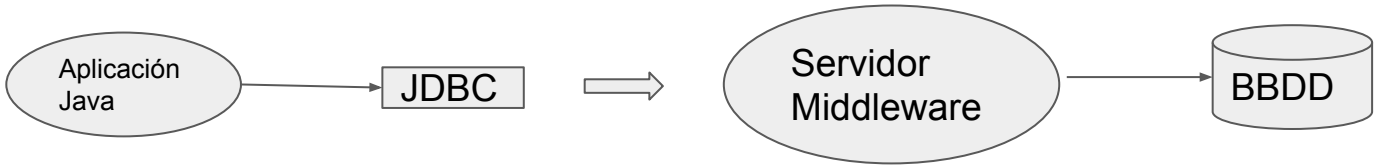
Tipos de drivers

- **Tipo 1 JDBC-ODBC Bridge**: permite el acceso a bases de datos JDBC mediante un driver ODBC. Convierte las llamadas al API de JDBC en ODBC. Exige la instalación y configuración de ODBC en la máquina cliente

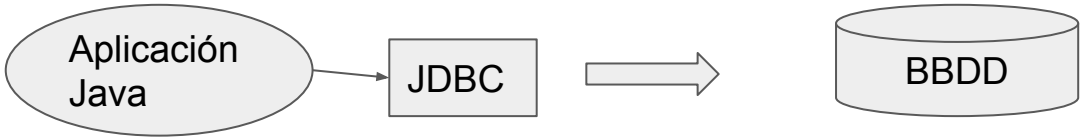


- **Tipo 2 Native**: controlador escrito parcialmente en Java y en código nativo de la base de datos. Traduce las llamadas al API de JDBC Java en llamadas propias del motor de base de datos. Exige instalar en la máquina cliente código binario propio del cliente de base de datos y del sistema operativo.

- **Tipo 3 Network:** controlador de Java puro que utiliza un protocolo de red (por ejemplo HTTP) para comunicarse con el servidor de base de datos. Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red y a continuación son traducidas por un software intermedio (Middleware) al protocolo usado por la bbdd. No exige instalación en cliente.



- **Tipo 4 Thin:** controlador de Java puro con protocolo nativo. Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red usado por el motor de base de datos. No exige instalación en cliente.



Los tipos 3 y 4 son la mejor forma de acceder. Los tipos 1 y 2 se usan normalmente cuando no queda otro remedio. En la mayoría de los casos la opción más adecuada será el tipo 4.

Cómo funciona JDBC

- JDBC define varias interfaces que permiten realizar operaciones con bases de datos; a partir de ellas se derivan las clases correspondientes.
- Estas clases están definidas en el paquete **java.sql**
- El funcionamiento de un programa con JDBC requiere los siguientes pasos:
  - Importar las clases necesarias
  - Cargar el **driver** JDBC
  - Identificar el origen de datos
  - Crear un objeto **Connection**
  - Crear un objeto **Statement**
  - Ejecutar una consulta con el objeto **Statement**
  - Recuperar los datos del objeto **ResultSet**
  - Liberar sucesivamente **ResultSet, Statement, Connection**

Ejemplo: acceso a MySql

Preparación máquina

- Clona la máquina Debian_MVJava. Llama a la nueva máquina Debian_MySql.
- Inicia la nueva máquina.
- Instala MySql: apt-get install mysql-client mysql-server
- Añade la bbdd “ejemplo”. Puedes utilizar el siguiente script sql.
- Descarga el conector JDBC para MySql desde aquí
- Añade en *eclipse* una librería externa a un nuevo proyecto (*properties* → *libraries* → *Add external jar*) que apunte al “.jar” recién descargado
- Copia el siguiente código en Debian_MySql y ejecútalo.

```
import java.sql.*;
public class Main {
	public static void main(String[] args) {
		try{
			Class.forName("com.mysql.cj.jdbc.Driver");
			Connection conexion=DriverManager.getConnection
("jdbc:mysql://localhost/ejemplo","austria","austria");
			Statement sentencia =conexion.createStatement();
			String sql = "SELECT * from depart";
			ResultSet result = sentencia.executeQuery(sql);

			while (result.next()){
				System.out.printf("%d, %s, %s, %n",
					result.getInt(1),
					result.getString(2),
					result.getString(3));
			}
			result.close();
			sentencia.close();
			conexion.close();
		} catch (ClassNotFoundException cn) { cn.printStackTrace();
		} catch (SQLException e) {e.printStackTrace();
		}
	}
}
```

Análisis del ejemplo anterior

Cargar driver

En primer lugar se carga el driver con el método **forName()** de la clase **Class** (java.lang). Para ello se le pasa un objeto String con el nombre de la clase del driver como argumento. En el ejemplo, como se accede a una base de datos MySql, necesitamos cargar el driver **com.mysql.jdbc.Driver:**
Class.forName("com.mysql.jdbc.Driver");

Establecer conexión

A continuación se establece la conexión con la base de datos. El servidor MySql debe estar rodando. Usamos la clase **DriverManager** con el método **getConnection()** de la siguiente manera:

```
Connection conexion=DriverManager.getConnection ("jdbc:mysql://localhost/ejemplo","austria","austria");
```

La sintaxis del método **getConnection()** es la siguiente:

```
public static Connection getConnection (String url, String user, String password) throws SQLException
```

El primer parámetro del método getConnection() representa la URL de conexión a la bbdd. Tiene el siguiente formato para conectarse a MySql:

```
jdbc:mysql://nombre_host:puerto/nombre_basedatos
```

- **jdbc:mysql** → indica que estamos utilizando un driver JDBC para MySql
- **nombre_host** → indica el nombre del servidor donde está la base de datos. Aquí puede ponerse una IP, el nombre de máquina, o localhost
- **puerto** → puerto donde está escuchando el servidor MySql. Por defecto es el 3306. Si no se indica se asume que es este valor.
- **nombre_basedatos** → nombre de la base de datos a la que queremos conectarnos. Debe existir previamente en MySql.

Ejecutar sentencias SQL

A continuación se realiza la consulta, para ello recurrimos a la interfaz **Statement** para crear una sentencia. Para obtener un objeto **Statement** se llama al método **createStatement()** de un objeto **Connection** válido. La sentencia obtenida (o objeto obtenido) tiene el método **executeQuery()** que sirve para realizar una consulta en la base de datos, se le pasa un String en que está la consulta SQL:

```
Statement sentencia =conexion.createStatement();
String sql = "SELECT * from departamentos";
ResultSet result = sentencia.executeQuery(sql);
```

El resultado nos lo devuelve como un **ResultSet,** que es un objeto similar a una lista en la que está el resultado de la consulta. Cada elemento de la lista es uno de los registros de la tabla “departamentos”. Recorremos la lista mediante el método **next():**

```
while (result.next()){
                    System.out.printf("%d, %s, %s, %n",
                        result.getInt(1),
                        result.getString(2),
                        result.getString(3));
                }
```

- Los métodos **getInt()** y **getString()** nos van devolviendo los valores de los campos de cada registro. Entre paréntesis se pone la posición de la columna en la tabla. También se puede poner una cadena que indica el nombre de la columna:

```
while (result.next()){
                    System.out.printf("%d, %s, %s, %n",
                        result.getInt(“dept_no”),
                        result.getString(“dnombre”),
                        result.getString(“loc”));
                }
```

- **ResultSet** dispone de varios métodos para mover el puntero que apunta a cada uno de los registros devueltos por la consulta:
  - **boolean next()** → Mueve el puntero una fila hacia adelante a partir de la posición actual. Devuelve **true** si el puntero se posiciona correctamente y **false** si no hay registros en **ResultSet.**
  - **boolean first()** → Mueve el puntero al primer registro de la lista.
  - **boolean last()** → Mueve el puntero al último registro de la lista.
  - **boolean previous()** → Mueve el puntero al registro anterior de la posición actual.
  - **void beforeFirst()** → Mueve el puntero justo antes del primer registro.
  - **int getRow()** → Devuelve el número de registro actual. Para el primer registro devuelve 1, para el segundo 2 y así sucesivamente.
- Por último, se liberan todos los recursos y se cierra la conexión

```
result.close();
sentencia.close();
conexion.close();
```

Problemas JDBC_1

1. Tomando como base el programa que ilustra los pasos del funcionamiento de JDBC obtén el APELLIDO, OFICIO y SALARIO de los empleados del departamento 10
2. Realiza otro programa Java utilizando la base de datos “ejemplo” que visualice el APELLIDO del empleado con máximo salario, visualiza también su SALARIO y el nombre del departamento.
3. Realiza un programa que busque los departamentos de una localidad. El programa solicitará el nombre de una localidad al usuario y devolverá los departamentos asociados a dicha localidad y los empleados de dicho departamentos.