UCH UNIVERSIDAD DE CIENCIAS Y HUMANIDADES

DESARROLLO DE SOFTWARE I

Eric Gustavo Coronel Castillo
gcoronelc.blogspot.com
ecoronel@uch.edu.pe

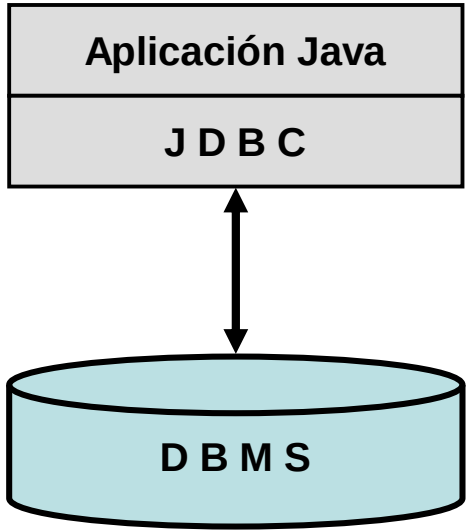
TEMA: JDBC – Parte I

Índice

- Objetivo
- JDBC
- Drivers JDBC
- Componentes del API JDBC
- Cargar el Driver JDBC
- Objeto Connection
- Acceso a una Instancia Única del Objeto Connection
- Objeto Statement
- Objeto ResultSet
- Objeto PreparedStatement

Objetivo

- Desarrollar aplicaciones que accedan a bases de datos utilizando el API JDBC.



Máquina Cliente

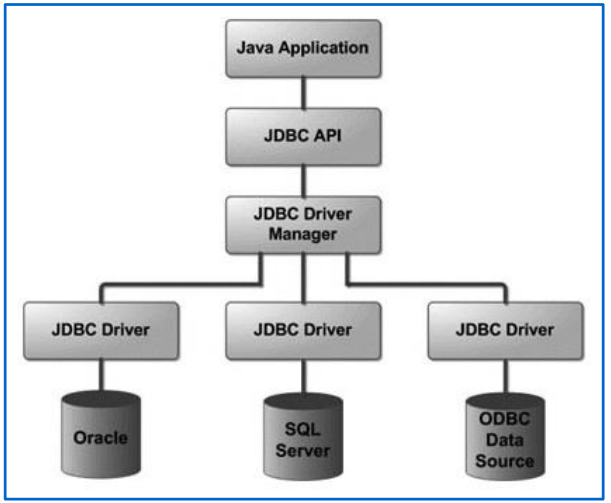
Protocolo
Propietario de Base
de Datos

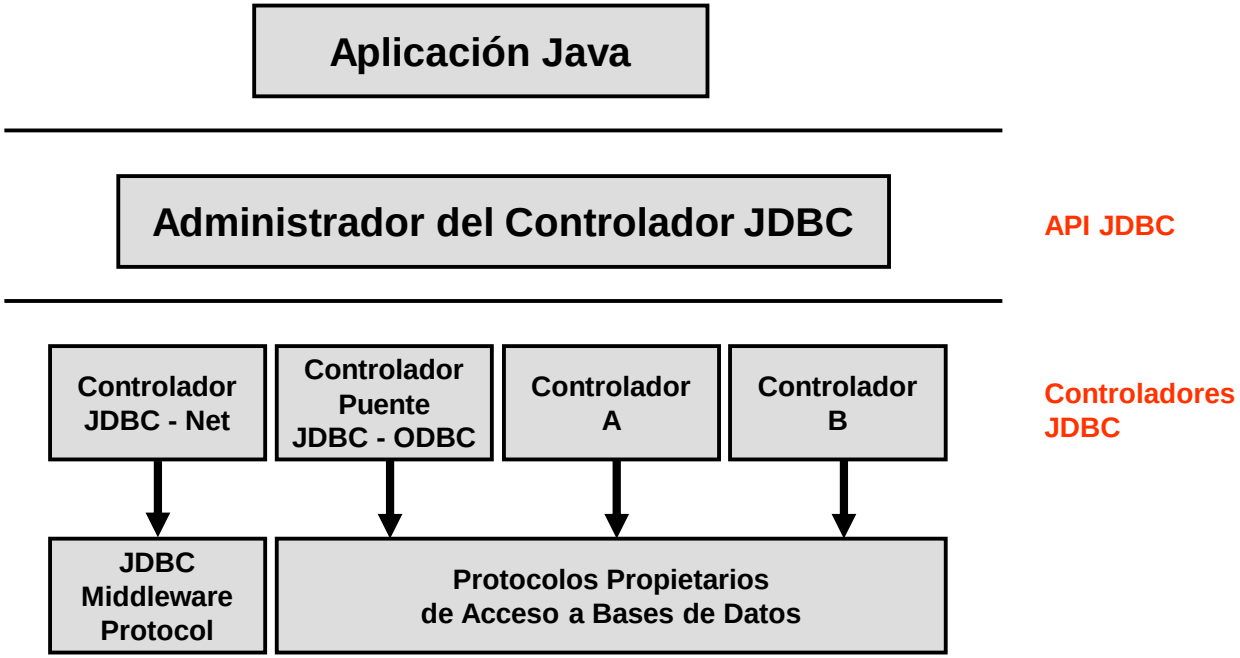
Servidor de
Base de Datos

JDBC

- Es la sigla de Java Database Connectivity.
- Es un API conformada por un conjunto de interfaces y clases Java que nos permiten acceder de una forma genérica a las bases de datos independiente del proveedor.
- Cada proveedor dispondrá de una implementación para comunicarse con su motor de base de datos.
- Se encuentra en el paquete java.sql.

- Básicamente una aplicación que usa JDBC realiza los siguientes pasos:
  - Establece una conexión con la base de datos.
  - Crea y envía una sentencia SQL a la base de datos.
  - Procesa el resultado.
  - Cierra la conexión.

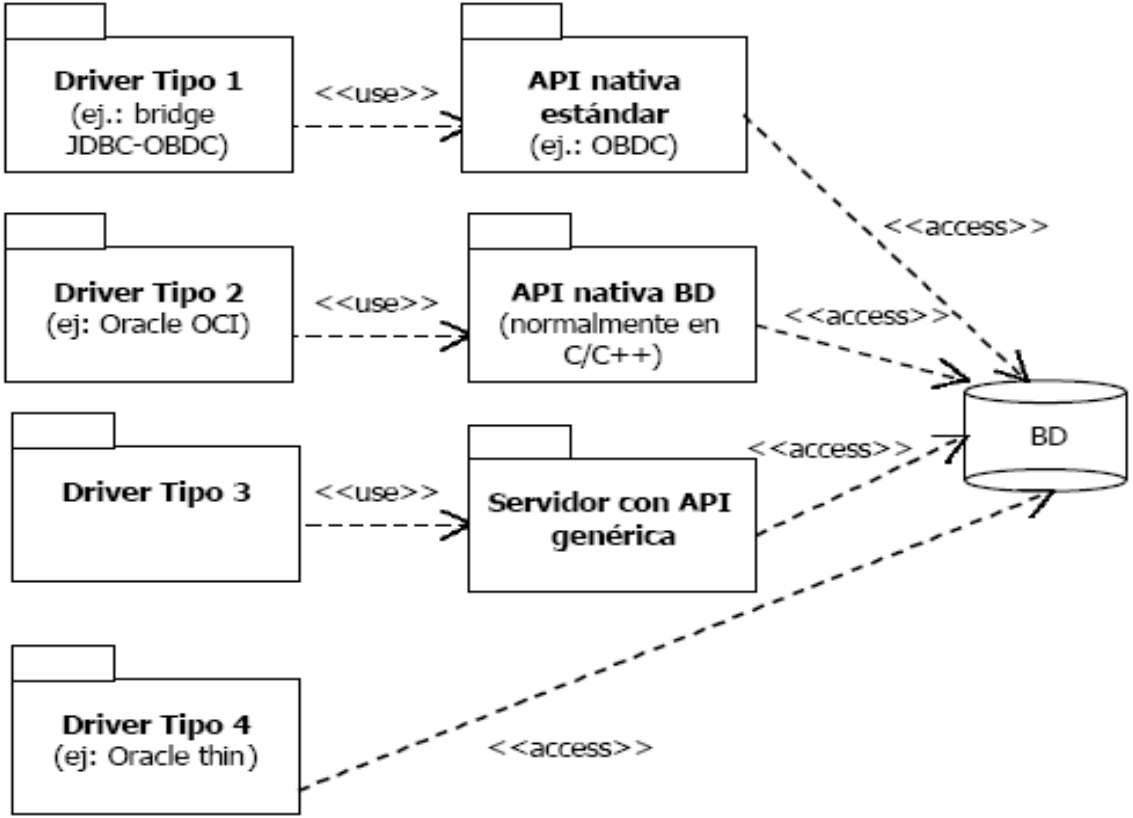



Aplicación Java
Administrador del Controlador JDBC
API JDBC
Controlador JDBC - Net
Controlador Puente JDBC - ODBC
Controlador A
Controlador B
Controladores JDBC
JDBC Middleware Protocol
Protocolos Propietarios de Acceso a Bases de Datos

Drivers JDBC

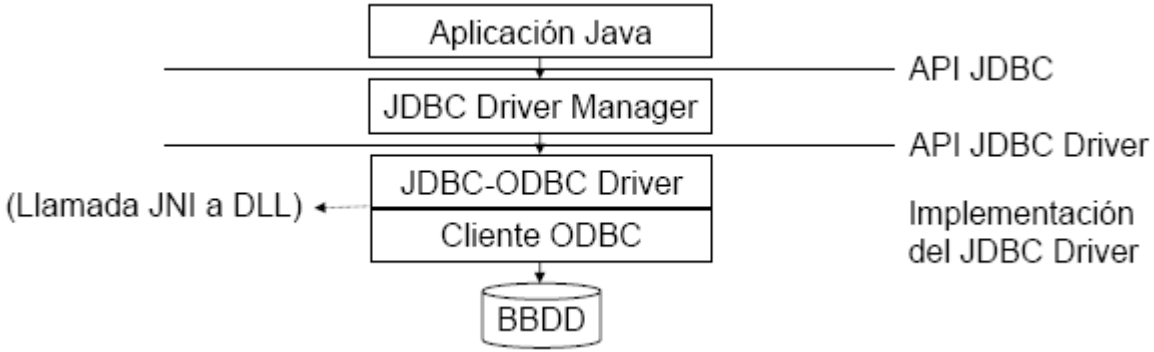
- Los drivers JDBC son la implementación que cada proveedor ha realizado del API JDBC.
- Existen cuatro tipos:
  - Tipo 1: JDBC - ODBC Bridge
  - Tipo 2: Native - API partly - Java
  - Tipo 3: JDBC - Net pure Java
  - Tipo 4: Native - Protocol pure Java
- Los SGBD tendrán un fichero JAR ó ZIP con las clases del driver JDBC que habrá que añadir a la variable CLASSPATH del sistema.
- Sun proporciona un driver JDBC-ODBC que permite el acceso a las fuentes de datos ODBC, como Microsoft Access, aunque no recomienda su uso en aplicaciones finales.

Drivers JDBC



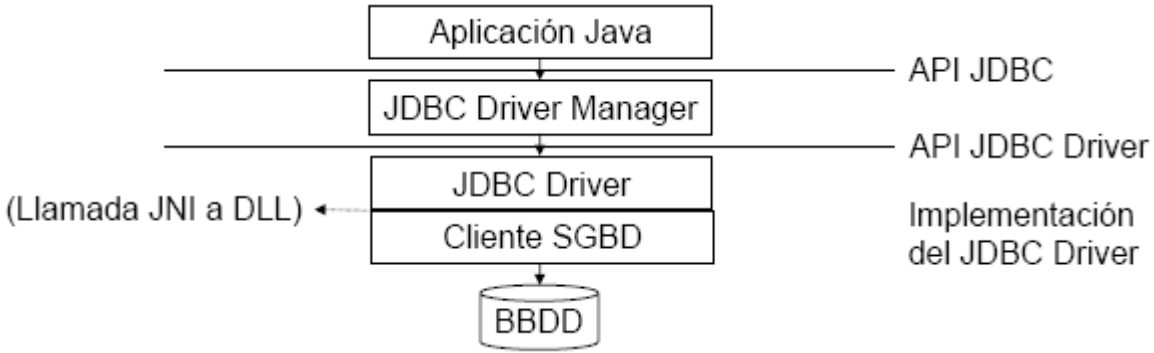
Drivers JDBC

- Tipo 1: JDBC - ODBC Bridge
  - Viene incluido con el JDK.
    - sun.jdbc.odbc.JdbcOdbcDriver
  - Traduce llamadas JDBC en llamadas ODBC.
  - Requiere de la instalación y configuración del cliente ODBC.



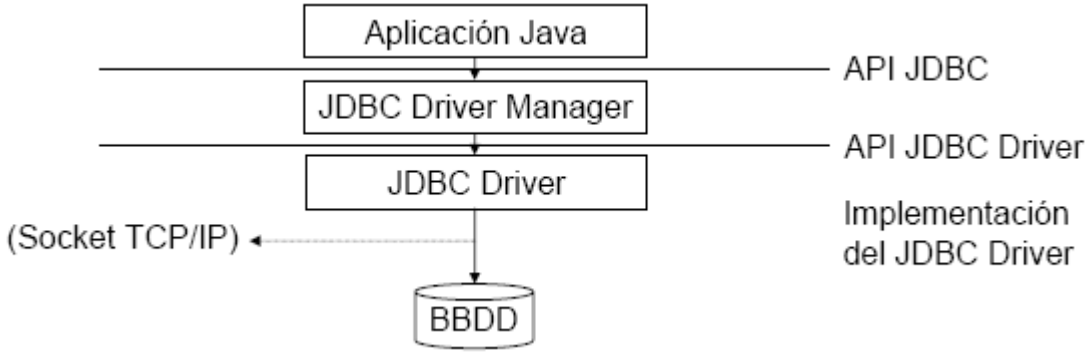
Drivers JDBC

- Tipo 2: Native - API partly - Java
  - No viene incluido con el JDK.
  - Traduce llamadas JDBC a llamadas propietarias del SGBD.
  - Requiere instalación y configuración del cliente del SGBD.



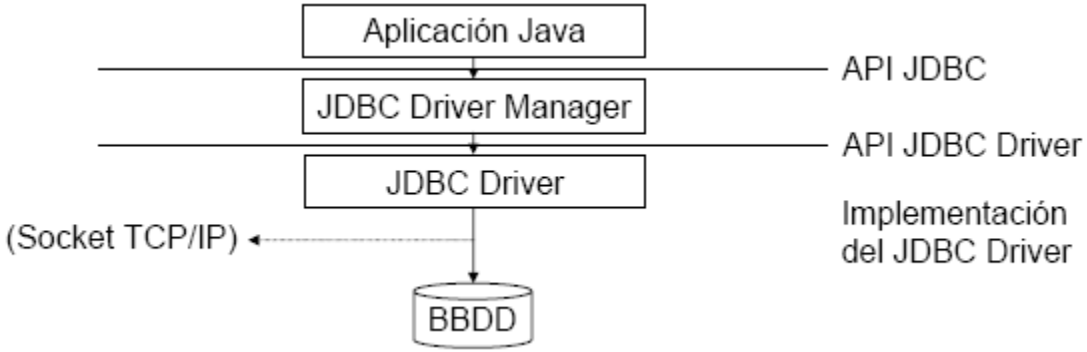
Drivers JDBC

- Tipo 3: JDBC - Net Pure Java
  - No viene incluido con el JDK
  - Conecta de manera remota vía TCP/IP con un daemon (listener) del SGBD (local o remoto).
  - El daemon traduce las llamadas al SGBD.
  - No requiere ninguna instalación previa.

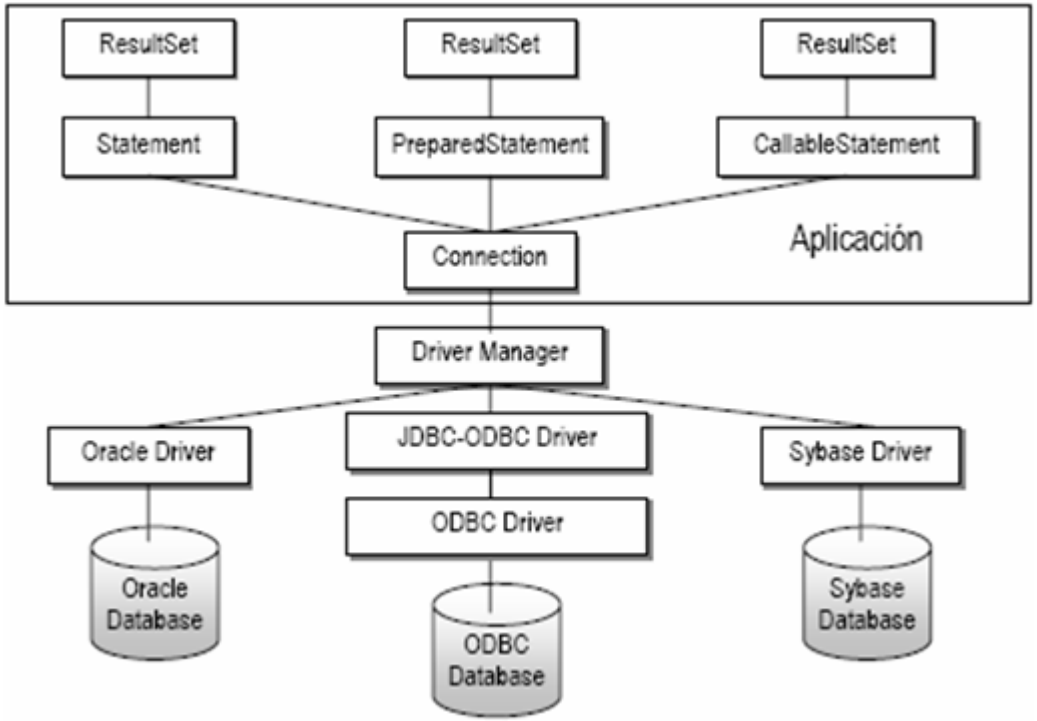


Drivers JDBC

- Tipo 4: Native - Protocol Pure Java
  - No viene incluido con el JDK
  - Conecta de manera remota vía TCP/IP con el SGBD (local o remoto).
  - No requiere ninguna instalación previa.



Componentes del API JDBC



Componentes del API JDBC

- Los componentes del API JDBC son:
  - Gestor de Drivers: java.sql.DriverManager
  - Conexión con la base de datos: java.sql.Connection
  - Ejecutar sentencias: java.sql.Statement
  - Manejo de resultado: java.sql.ResultSet
  - Sentencias con parámetros: java.sql.PreparedStatement
  - Procedimiento almacenado: java.sql.CallableStatement

Cargar el Driver

```
try {

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

} catch (ClassNotFoundException e) {

   System.out.println("Error loading driver: “ +
   e.getMessage());

}
```

Objeto Connection

- Definir la URL de Conexión de BD

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

- Establecer la Conexión

```
try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String url = "jdbc:mysql://localhost:3306/eurekabank";
        Connection cn = DriverManager.getConnection(url,"root",“admin");
} catch (Exception e) {
        System.out.println("Error loading driver: “ + e.getMessage());
}
```

- Cerrar la Conexión

```
cn.close();
```

Objeto Connection

- Obteniendo información del DBMS

```
try {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      String url = "jdbc:mysql://localhost:3306/eurekabank";
      Connection cn = DriverManager.getConnection(url,"root","admin");
      DatabaseMetaData dbmd = cn.getMetaData();
      String dbms = dbmd.getDatabaseProductName();
      String version = dbmd.getDatabaseProductVersion();
      System.out.println("Database: " + dbms);
      System.out.println("Version: " + version );
} catch (Exception e) {
      System.out.println(e.getMessage());
}
```

Acceso a una Instancia del Objeto Connection

```
public final class AccesoDB {

  private AccesoDB() {
  }

  public static Connection getConnection() throws SQLException {
    Connection cn = null;
    try {
      Class.forName("oracle.jdbc.OracleDriver").newInstance();
      String url = "jdbc:oracle:thin:@localhost:1521:XE";
      cn = DriverManager.getConnection(url, "eureka", "admin");
    } catch (SQLException e) {
      throw e;
    } catch (ClassNotFoundException e) {
      throw new SQLException("No se encontró el driver de la base de datos.");
    } catch (Exception e) {
      throw new SQLException("No se puede establecer la conexión con la BD.");
    }
    return cn;
  }

}
```

Objeto Statement

- Creando un Statement

```
Statement stm = cn.createStatement();
```

- Ejecutando una consulta

```
String query = "select vch_cliepaterno,vch_cliematerno," +
      "vch_clienombre from cliente";
ResultSet rs = stm.executeQuery(query);
```

  - Para modificar la BD, use executeUpdate, pasando un argumento que contenga UPDATE, INSERT o DELETE.
  - Use setQueryTimeout para especificar un tiempo de espera por resultados.

Objeto ResultSet

- Procesando Resultados

```
ResultSet rs = stm.executeQuery(query);
while( rs.next() ){
    System.out.println(rs.getString("vch_cliepaterno") +
        " " + rs.getString("vch_cliematerno") +
        " " + rs.getString("vch_clienombre") );
}
```

  - Primera columna tiene índice 1, no 0.
  - ResultSet provee varios métodos getXxxx que toman el índice o nombre de la columna a devolver el dato.

Objeto PreparedStatement

- Permite ejecutar sentencias SQL precompiladas.
- Podemos definir parámetros de entrada.
- Cada parámetro de entrada está definido por un signo de interrogación (?).
- Antes de ejecutarse la sentencia se debe especificar un valor para cada uno de los parámetros a través de los mtodos setXXX apropiados.

PreparedStatement

- Ejemplo

```
String sql = "select * from cliente where vch_cliedireccion like ?";
PreparedStatement ps = cn.prepareStatement(sql);
ps.setString(1, "%Lince%");
ResultSet rs = ps.executeQuery();
```

Enlaces

- http://gcoronelc.blogspot.pe/2017/02/seminario-acceso-base-de-datos-con-jdbc.html
- http://gcoronelc.blogspot.pe/2015/01/java-jdbc-resultset-to-list.html
- http://gcoronelc.blogspot.pe/2015/01/java-jdbc-resultset-to-list.html
- http://gcoronelc.blogspot.pe/2017/02/seminario-acceso-base-de-datos-con-jdbc.html
- http://gcoronelc.blogspot.pe/2014/05/java-jdbc-oracle-cursor.html
- http://gcoronelc.blogspot.pe/2017/08/separata-de-java-cliente-servidor.html

UCH UNIVERSIDAD DE CIENCIAS Y HUMANIDADES

Eric Gustavo Coronel Castillo
gcoronelc.blogspot.com
ecoronel@uch.edu.pe