Java Database Connectivity

M. en C. José Asunción Enríquez Zárate

Índice

- Introducción
- Accesos básicos
- Tipos SQL y Java

Introducción

- JDBC (Java DataBase Connectivity) es un API que permite lanzar queries a una base de datos.
- Su diseño está inspirado en dos conocidas APIs
  - ODBC (Open DataBase Connectivity)
  - X/OPEN SQL CLI (Call Level Interface)

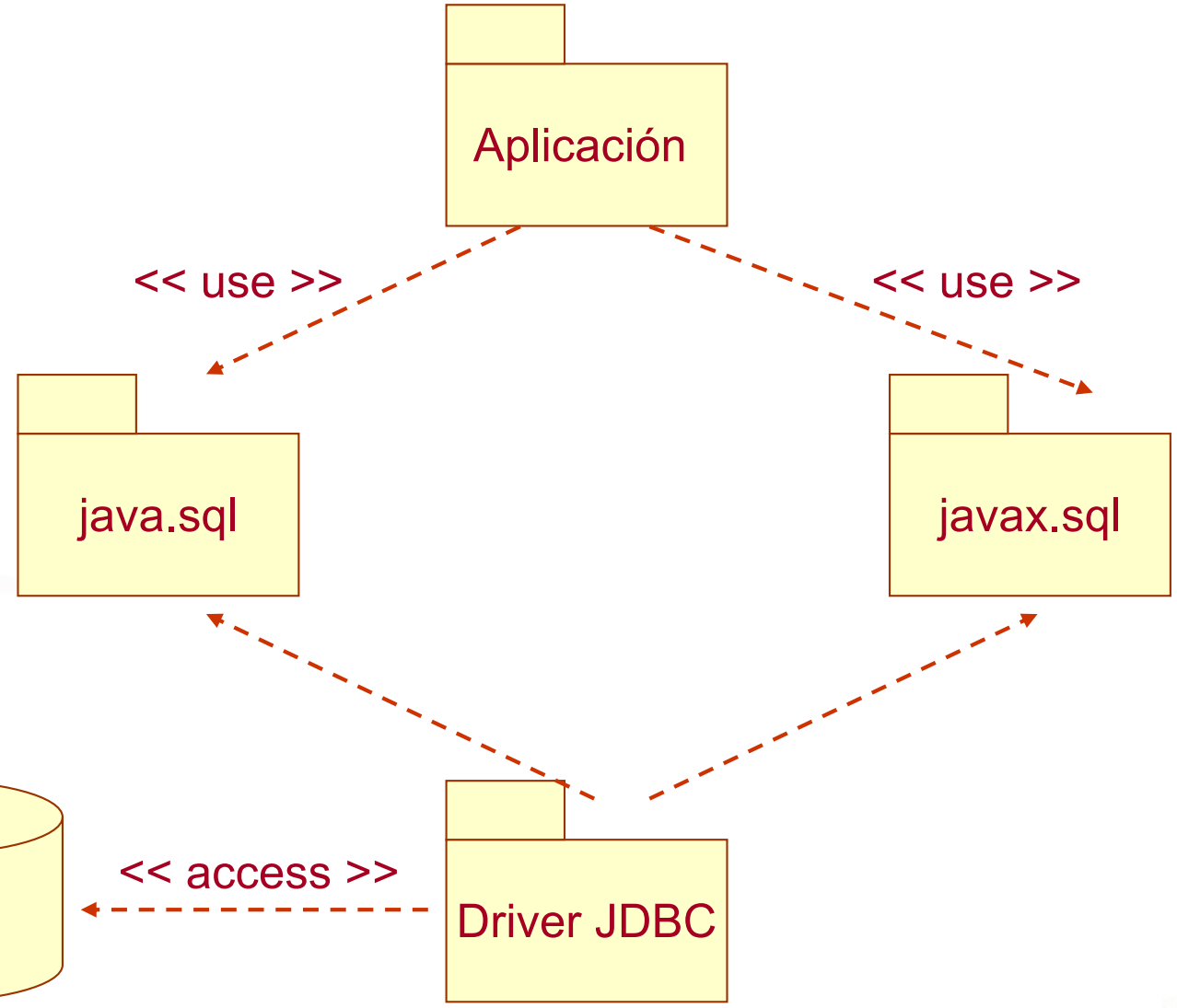
Introducción

- El programador siempre trabaja contra los paquetes **java.sql** y **javax.sql.**
  - Forma parte de JSE.
    - **javax.sql** formaba parte de JEE, pero se movió a JSE (desde la versión 1.4 de JSE).
  - Contienen interfaces y algunas clases concretas, que conforman el API.

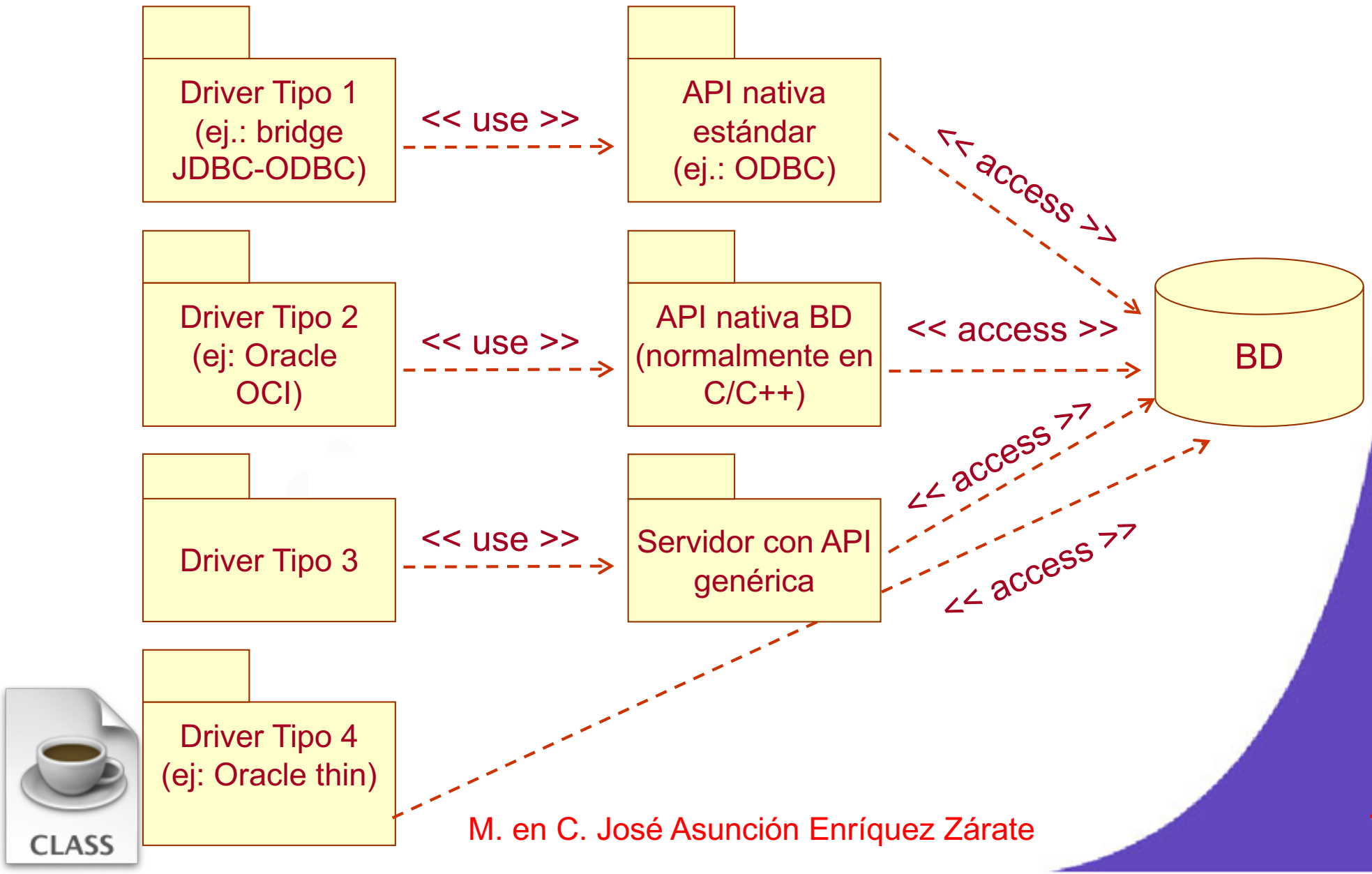
Introducción

- Para poder conectarse a la BD y lanzar queries, es preciso tener un driver adecuado.
  - Un driver suele ser un fichero **.jar** que contiene una implementación de todas las interfaces del API.
  - El código nunca depende del driver, dado que siempre trabaja contra los paquetes **java.sql** y **javax.sql.**

Driver JDBC



Tipos de drivers



Comentarios.

- Los drivers de tipo 1 y 2 llaman a APIs nativas mediante JNI (Java Native Interface)
  - Requieren que la máquina en la que corre la aplicación tenga instaladas las librerías de las APIs nativas (.DLL, .so).
- Los drivers de tipo 3 y 4 son drivers 100% Java.

Comentarios.

- La ventaja de un driver de tipo 3 es que una aplicación puede usarlo para conectarse a varias BDs.
  - Pero esto también se puede conseguir usando un driver distinto (de los otros tipos) para cada BD.

Independencia de la BD

- Idealmente, si la aplicación cambia de BD, no necesitamos cambiar el código; simplemente, necesitamos otro driver.
- Sin embargo, desafortunadamente las BDs relacionales usan distintos dialectos de SQL (A pesar de que en teoría es un estándar).
  - Tipos de datos: varían mucho según la BD.
  - Generación de identificadores: secuencias, autonumerados, etc.

Proceso de Carga JDBC

- Cargar el driver con DriverManager.
- Obtener un objeto Connection mediante DriverManager.getConnection
  - conexion = DriverManager.getConnection (urlBaseDeDatos, usuario, contrasena);
- Obtener un objeto Statement mediante createStatement del objeto Connection
  - statement = conexion.createStatement();

Proceso de Carga JDBC

- Obtenga un objeto ResultSet mediante executeQuery del objeto Statement
  - tablaCategoriaArticulos = statement.executeQuery(consulta);
- Ejecute una iteración sobre el ResultSet
  - while (tablaCategoriaArticulos.next()) {

Proceso de Carga JDBC

- Cerrar el objeto ResultSet
- Cerrar el objeto Statement
- Cerrar el objeto Connection

Ejemplo de actualización Insertar Registros

```
package com.darkdestiny.jdbctutorial;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.SQLException;
/**
 *
 * @author Jose Asuncion
 */
```

```
public class EjemploInsert {
    /** Creates a new instance of EjemploInsert */
    public EjemploInsert() {}

    public static void main(String[] args) throws
    ClassNotFoundException {
        Connection conexion = null;
        Statement statement = null;
```

```
try {
        /* Obtener una Conexion. MySql 8+ */
      String driverMySql = "com.mysql.cj.jdbc.Driver";
        String urlBaseDeDatos =
        jdbc:mysql://localhost:3306/
EzjaCatalogo?serverTimezone=America/Mexico_City&
useSSL=false&allowPublicKeyRetrieval=true
        String usuario = "ezja";
        String contrasena = "ezja";
        Class.forName(driverMySql);
        conexion =
DriverManager.getConnection(urlBaseDeDatos,
usuario, contrasena);
```

```
/* Crear "statement". */
        statement = conexion.createStatement();
        // Insertar una nueva categoría a la base de datos.
        String consulta = "INSERT INTO articuloscategorias
" +
                "(Id, Descripcion) VALUES (4,'Deportes')";
        /* Ejecutar la Consulta. */
        int filasInsertadas =
statement.executeUpdate(consulta);
        if (filasInsertadas != 1) {
            throw new SQLException("Ploblemas durante la
insercion !!!!");
        }
```

```
else{
            System.out.println("El registro se inserto
    satisfactoriamente");
            }
        }catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (statement != null) {
                    statement.close();
                }
```

```
if (conexion != null) {
                    conexion.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo de Consulta Seleccion

```
package com.darkdestiny.jdbctutorial;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
/**
 *
 * @author Jose Asuncion
 */
```

```
public class EjemploSelect {
    /** Creates a new instance of EjemploSelect */
    public EjemploSelect() {
    }

    public static void main(String[] args) throws
    ClassNotFoundException{
        Connection conexion = null;
        Statement statement = null;
        ResultSet tablaCategoriaArticulos=null;
        try {
```

```
/* Obtener una Conexion. */
    << ……. >>

        /* Crear "statement". */
        statement = conexion.createStatement();
        /* Seleccionar todos los registros de la tabla
  categoría. */
        String consulta = "select * from articuloscategorias";
        /* Ejecutar la Consulta. */
        tablaCategoriaArticulos =
  statement.executeQuery(consulta);
```

```
/* Mostrar todos los encabezados  */
        for (int x = 1; x <=
   tablaCategoriaArticulos.getMetaData().getColumnCount(
   ); x++) {
            System.out.print(" |" +
   tablaCategoriaArticulos.getMetaData().getColumnName(
   x)+ " | " );
        }
        System.out.println();
```

```
/* Iterar sobre los resultados. */
        while (tablaCategoriaArticulos.next()) {
            int i = 1;
            int id = tablaCategoriaArticulos.getInt(i++);
            String descripcion =
  tablaCategoriaArticulos.getString(i++);
            System.out.println(" | " + id + " | " + descripcion +
  "   |  ");
        }

    }catch (SQLException e) {
        e.printStackTrace();
    }
```

```
finally {
        try {
            if (statement != null) {
                statement.close();
            }
            if (conexion != null) {
                conexion.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } } }
```

Comentarios

- driverMySql = "com.mysql.cj.jdbc.Driver"
  - Atributo de tipo String que almacena el nombre del controlador JDBC para MySql versión 8 +.
- urlBaseDeDatos = "jdbc:mysql://localhost/EzjaCatalogo"
  - Atributo de tipo String que almacena el nombre y ruta donde se encuentra almacenada la base de datos, puede ser una dirección Ip(ej. jdbc:mysql://127.0.0.1:3306/EzjaCatalogo).

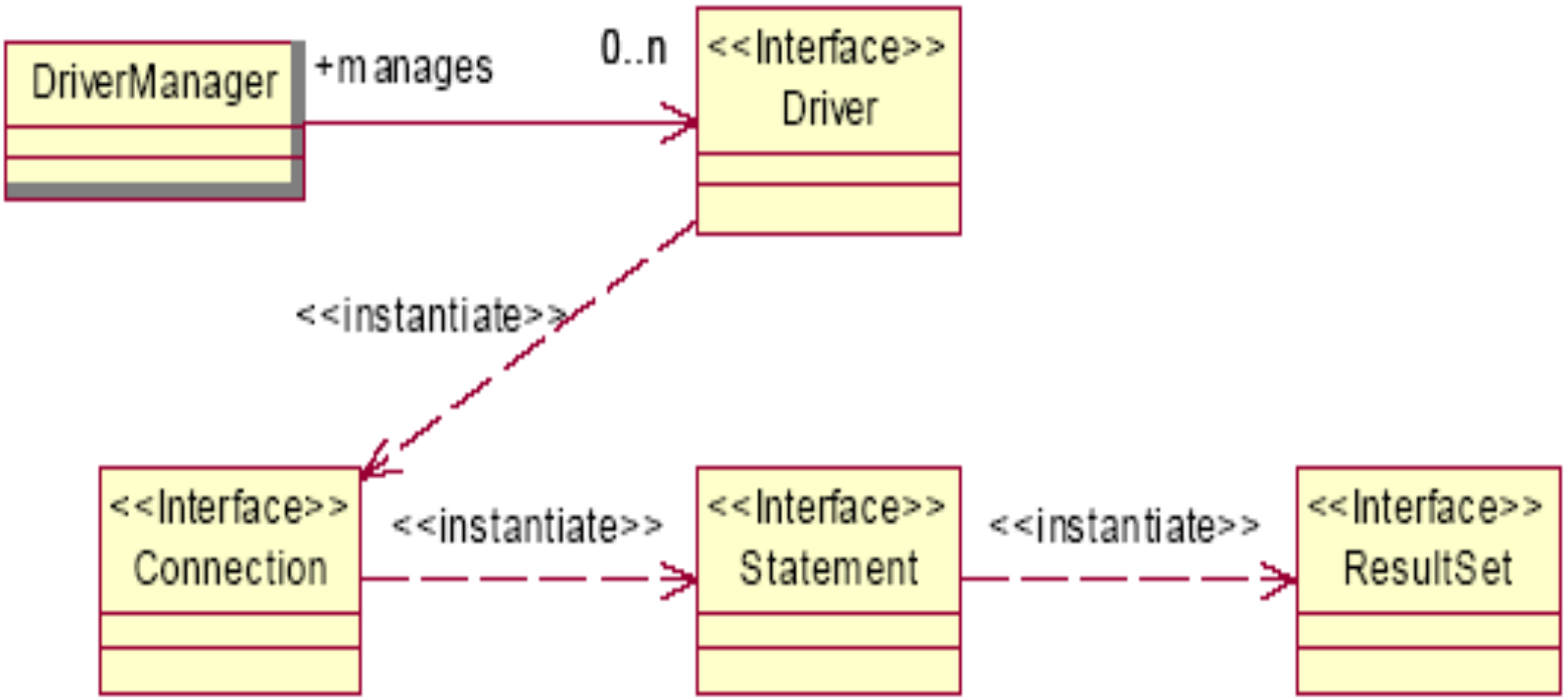
Comentarios

- ?serverTimezone=America/Mexico_City&useSSL=false&allowPublicKeyRetrieval=true
  - Parametros para realizar la correcta conexión con MySql en versiones 8 y superiores.
- usuario = "ezja"
  - Atributo de tipo String que contiene el nombre del usuario de la base de datos.
- contrasena = "ezja".
  - Atributo que almacena el password del usuario de la base de datos.

Comentarios

- Class.forName(driverMySql).
  - Registra la clase en el cargador de clases.
- DriverManager.getConnection(urlBaseDeDatos, usuario, contrasena)
  - Obtiene una conexión a la base de datos a partir de su url, usuario y contraseña.

Comentarios



Liberación de recursos

- En principio, aunque no se llame a Connection.close, uando la conexión sea eliminada por el garbage collector, el método finalize de la clase que implementa Connection, invocará al método close
- Además
  - Cuando se cierra una conexión, cierra todos sus **Statements** asociados.
  - Cuando se cierra un **Statement**, cierra todos sus **ResultSet**s asociados.

Liberación de recursos

- Sin embargo,
  - En una aplicación multi-thread que solicita muchas conexiones por minuto (ej.: una aplicación Internet), el garbage collector correrá demasiado tarde => **es imprescindible cerrar las conexiones tan pronto como se pueda.**
  - Puede haber bugs en algunos drivers, de manera que no cierren los **Statement**s asociados a una conexión y los **ResultSet**s asociados a un **Statement** => **mejor cerrarlos explícitamente**

PreparedStatement

- Cada vez que se envía una query a la BD, ésta construye un plan para ejecutarla.
  - La parsea.
  - Determina qué se quiere hacer.
  - Determina cómo ejecutarla.
- Permite representar una query parametrizada, para la que la BD construirá un plan.

Statement vs PreparedStatement

- **PreparedStatement** es más eficiente en bucles que lanzan la misma query con distintos parámetros.
- **Statement** obliga a formatear los datos de la Query porque ésta se envía a la BD tal cual se escribimos.
- Las cadenas de caracteres tienen que ir entre ‘’ => código menos legible y más propenso a errores.

Statement vs PreparedStatement

- Existen algunos tipos de datos (ej.: fechas) que se especifican de manera distinta según la BD.
- Con **PreparedStatement** el formateo de los datos lo hace el driver
- Conclusión
  - En general, usar siempre que sea posible **PreparedStatement.**

Tipos SQL y Java

- **ResultSet** y **PreparedStatement** proporcionan métodos **getXXX** y **setXXX**
  - ¿ Cuál es la correspondencia entre tipos Java y tipos SQL ?.
- Idea básica: un dato de tipo Java se puede almacenar en una columna cuyo tipo SQL sea consistente con el tipo Java.
- Las BDs suelen emplear nombres distintos para los tipos que proporcionan.

Correspondencia entre tipos Java y SQL estándar

Tipo Java	Tipo SQL
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
String	VARCHAR o LONGVARCHAR
byte[]	VARBINARY o LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP


Attribution-NonCommercial-
NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)


CLASS

Java Database Connectivity

M. en C. José Asunción Enríquez Zárate