Conceptos básicos de JDBC

Universidad de Oviedo

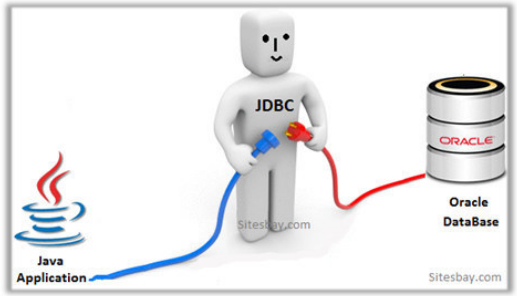
Repositorios de Información

2021-2022

JDBC: Conceptos y características

JDBC Definición

JDBC es una **API** que permite a programas **Java** acceder a un amplio rango de **SGBD** a través de **SQL**.



- Consiste en un conjunto de clases e interfaces Java que abstraen las funciones típicas de acceso a base de datos: realizar conexiones, enviar sentencias SQL, procesar los resultados, etc . . .
- Cada sistema gestor de base de datos proporciona su propia implementación a través de un **Controlador** o **Driver**.

Características de JDBC

Independencia de la base de datos

Los métodos son genéricos y la interfaz proporciona una librería de llamadas a funciones.

- Conexión estándar.
- Tipos de datos estándar.
- Códigos de error estándar.

Los controladores proporcionan diferentes implementaciones y son cargados y utilizados en tiempo de ejecución en el lado del cliente.

Características de JDBC

Independencia de la plataforma

Los controladores JDBC están desarrollados en Java y existen para casi cualquier sistema gestor de base de datos.

Mapeo objeto-relacional

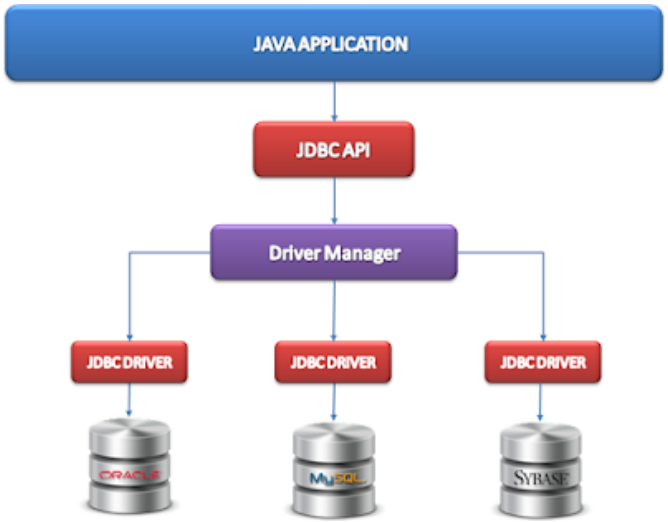
Conversión entre el sistema de tipos utilizado por cualquier LPOO y un SGBR.

"Virtualización"de una BBDD orientada a objetos sobre la relacional.

Proporciona optimización en la busqueda/indexación y flexibilidad en los procesos de ingeniería.

Arquitectura JDBC

Application Inicia y termina conexiones, ejecuta sentecias SQL, ... (**Java-based API**)



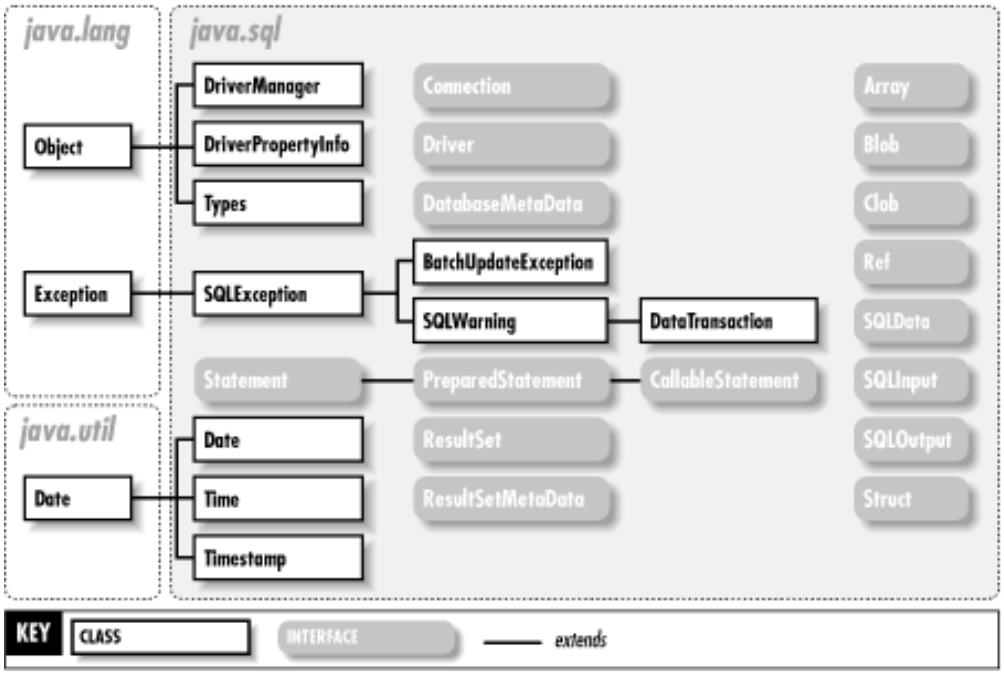
Para conectar a una BBDD necesitamos un **controlador JDBC especifico**[1] que nos proporciona el fabricante. Una clase que implemente la interfaz para la BBDD (com.dbProvider.jdbc.Driver) que proporciona la conexión y convierte las llamadas Java a instrucciones propias de la BBDD.

Driver Manager carga el driver en tiempo de ejecución.
Data source procesa las instrucciones SQL.

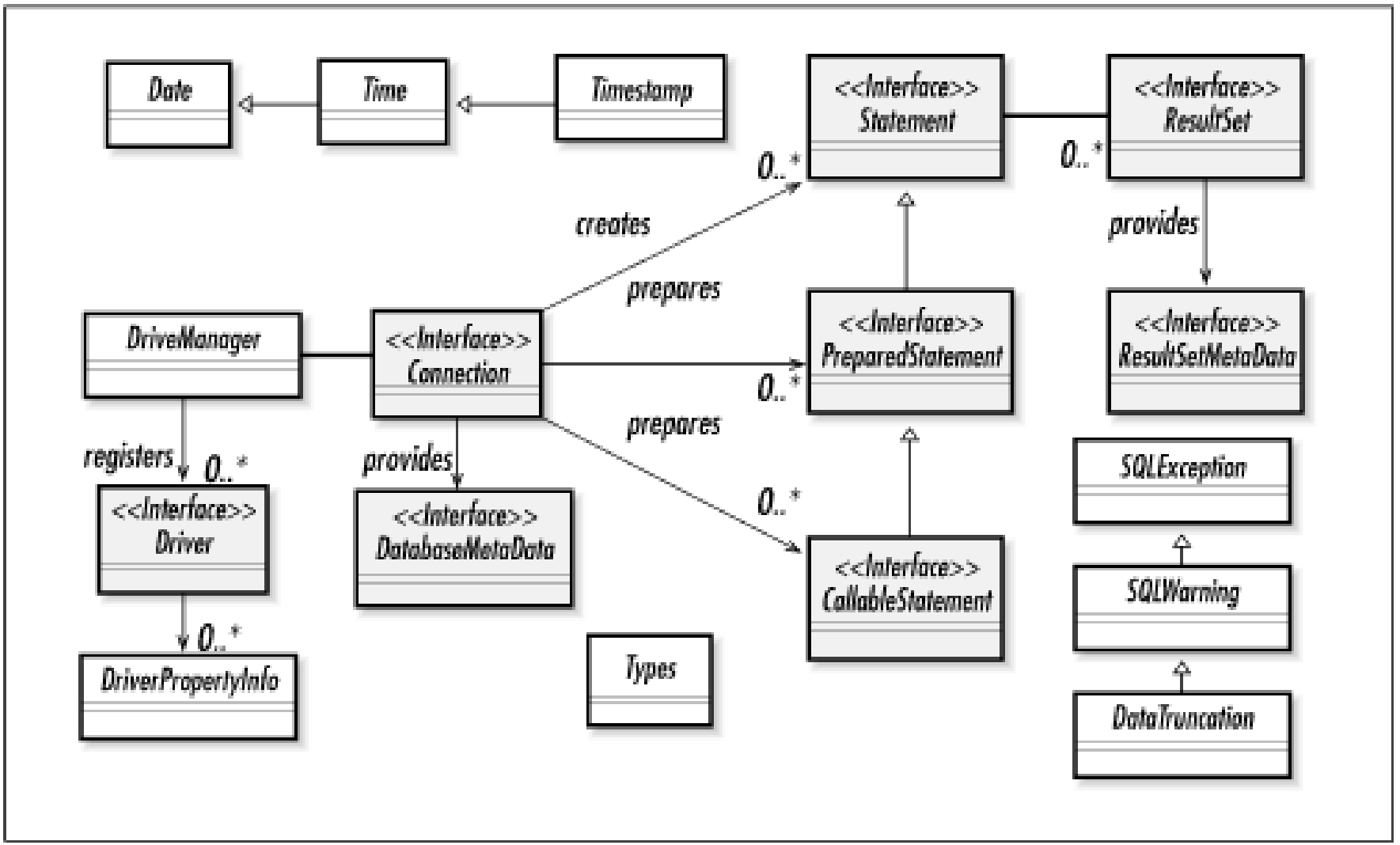
[1] `https://docs.oracle.com/javase/tutorial/jdbc/overview/architecture.html`

JDBC API

La API JDBC queda definida en dos paquetes principales.
`java.sql` **classes principales** para acceder a la BBDD.
`javax.sql` **características adicionales** para aplicaciones empresariales

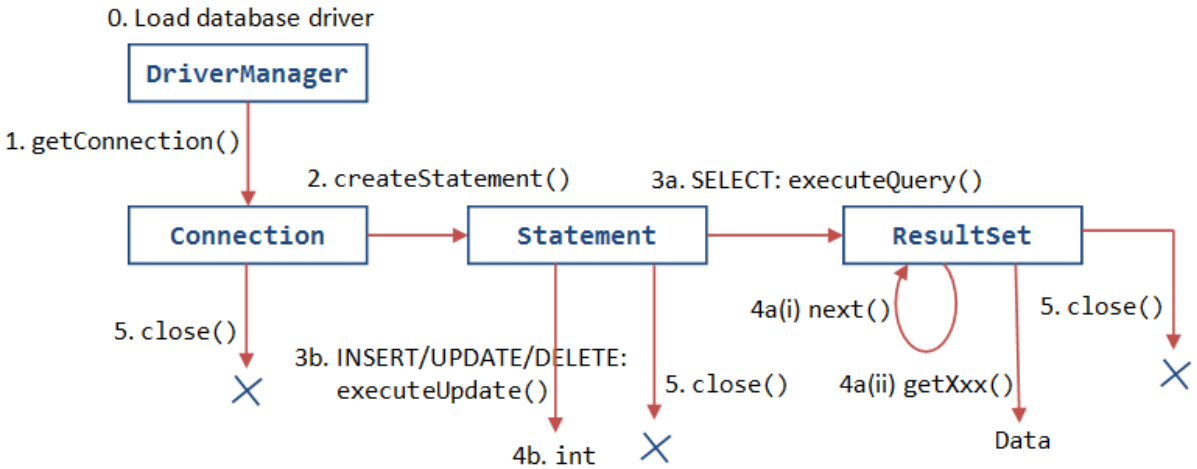


Principales clases e interfaces de JDBC



JDBC: Proceso de desarrollo

Proceso de desarrollo



Proceso de desarrollo

```
    private String URL = "jdbc:mysql://localhost:3306/DB";
    private String USER = "myuser";
    private String PASSWORD = "mypass";
    private Connection conn = null;
    private Statement stmnt = null;
    private ResultSet rs = null;

    public void doSomething () {
        try {
            conn = DriverManager.getConnection(URL, USER, PASSWORD);
            stmnt = conn.createStatement();
            rs = stmnt.executeQuery ( "SELECT a, b FROM table ");
            while ( rs.next() )
            {
                int a = rs.getInt( "a" );
                String b = rs.getString( "b" );
                System.out.println("a = " + a + " b = " + b);
                } //while
            }
        catch (SQLException sqle)
        {
            System.out.println("SQL Exception thrown: " + sqle);
        }
        finally {
            if ( rs != null) { try {rs.close(); } catch (SQLException e) {} }
            if ( stmnt != null) { try {stmnt.close(); } catch (SQLException e) {} }
            if ( conn != null) { try {conn.close(); } catch (SQLException e) {} }
            }
        }
}
```

Proceso de desarrollo

Antes de conectar con la BBDD

1. **URL**

   jdbc:protocol:subprotocol://host name:port/connection details

   `jdbc:mysql://127.0.0.1:3306/mydb`

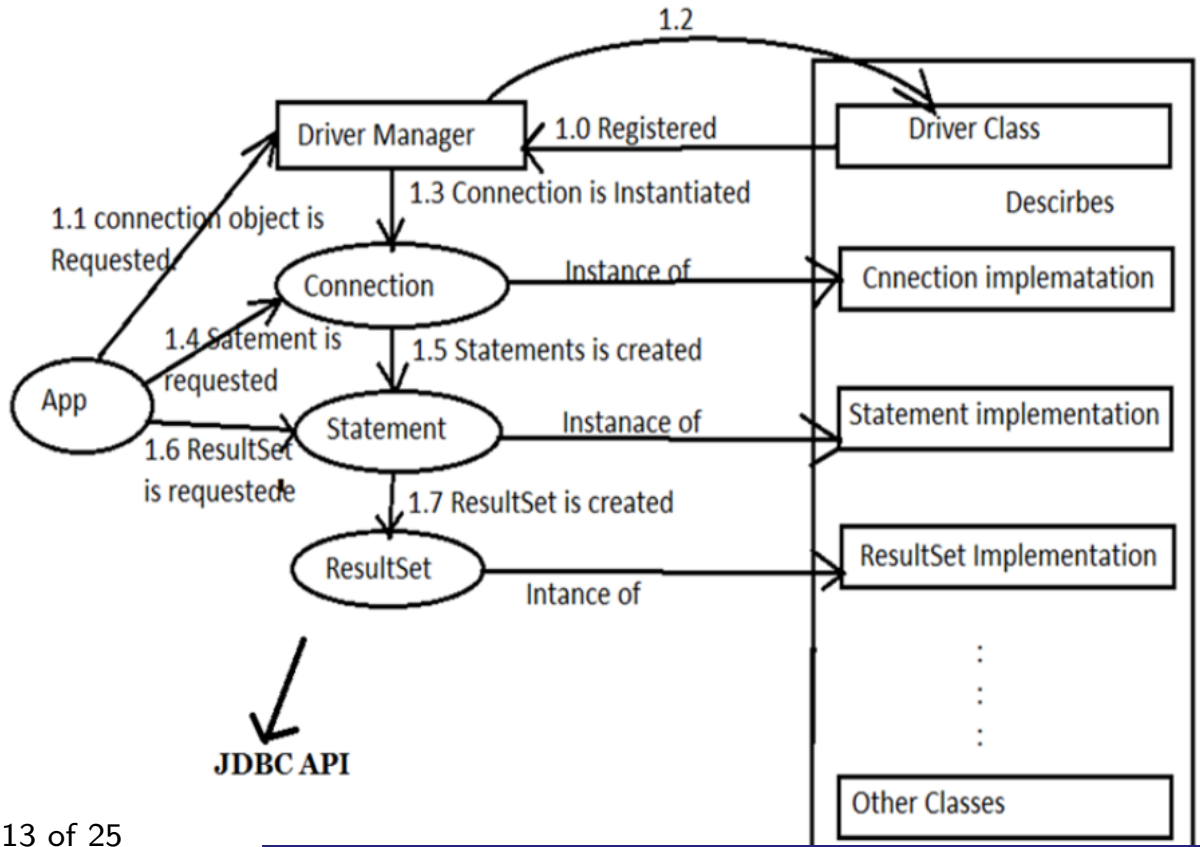
   `jdbc:oracle:thin:@localhost:1521:orcl`
2. **Credenciales: Nombre de usuario y contraseña**

   `jdbc:mysql://127.0.0.1:`
   `3306/mydb?user=me&password=psswd`
3. **Controlador**
   - 3.1 Interfaz (java.sql), con la especificación para implementar
   - 3.2 La clase del driver (`sun.jdbc.odbc.JdbcOdbcDriver`) que proporciona
   - 3.3 La implementation de las clases de interfaz de la API JDBC que convierte las llamadas en Java a llamadas especificas de la BBDD. `ojdbc6.jar`, `mysql-connector.jar` etc.

Proceso de desarrollo

Carga del controlador

El `DriverManager` envía los valores de los parámetros a cada controlador registrado hasta que encuentra uno que pueda establecer la conexión.



Proceso de desarrollo

Conexión

```
getConnection(String url, String user, String passwd)
getConnection(String url)
getConnection(String url, Properties info)
throws SQLException
```

```
try {
    conn = DriverManager.getConnection(URL, USER, PASSWORD);
    }
catch (SQLException sqle)
{
    System.out.println("SQL Exception thrown: " + sqle);
}
```

Proceso de desarrollo

Conexión

Interfaz de conexión

- Niveles de aislamiento
  `public int getTransactionIsolation()`
  `void setTransactionIsolation(int level)`
- Transacciones en modo sólo lectura
  `public boolean getReadOnly()`
  `void setReadOnly(boolean b)`
- Autocommit `public boolean getAutoCommit() void setAutoCommit(boolean b)`
- Estado de la conexión
  `public boolean isClosed()`

Proceso de desarrollo

Ejecución de instrucciones SQL

Para acceder a las tablas de la BBDD se utiliza la interfaz `Statement`.

```
Statement stmnt = conn.createStatement();
```

El objeto stmnt tendrá los siguientes métodos para ejecutar consultas SQL:

Method	Returns	Used for
`executeQuery(sqlString)`	`ResultSet`	`SELECT` statement
`executeUpdate(sqlString)`	`int` (rows affected)	`INSERT`, `UPDATE`, `DELETE`, or a DDL
`execute(sqlString)`	`boolean` (true if there was a `ResultSet`)	Any SQL command or commands

Proceso de desarrollo

Gestión de los resultados

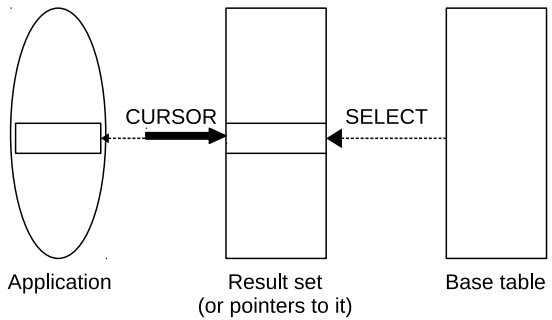
El conjunto de filas resultado de ejecutar la sentencia SQL son devueltas en un objeto que implementa la interfaz `java.sql.ResultSet`.

Problema: Buffer mismatch o Impedance Mismatch

- Las tablas en una BBDD pueden ser muy grandes, sin embargo un programa maneja buffers de tamaño fijo.

Solución: Cursor

- El `ResultSet` establece un cursor apuntando a la fila en la que se encuentra. Cuando es creado el cursor se posiciona justo antes de la primera fila y se va obteniendo una sola fila cada vez.



Proceso de desarrollo

Gestión de los resultados

- Para mover el cursor se usa `next()` y devuelve un valor lógico indicando si existe otra fila.
- Métodos para leer los datos: *`getXXX`*`(String fieldName)` *`getXXX`*`(int columnIndex`[1]`)`
  - `getInt()`, `getLong()`, `getString()`
  - La consulta devuelve tipos de datos SQL pero el controlador intenta convertirlos a tipos de datos propios de Java.

[1] El índice de la primera columna es 1, no 0

Proceso de desarrollo

Gestión de los resultados

Otras opciones de navegación

- `previous()` Navega a la fila anterior.
- `first()` Navega a la primera fila.
- `last()` Navega a la última fila.
- `beforeFirst()` Si ejecutamos `next` después de llamar a este método, devuelve la primera fila.
- `afterLast()` Si ejecutamos `previous` después de llamar a este método, devuelve la última fila.
- `relative(int numOfRows)` Navega hacia adelante o hacia atrás un número de filas especificado desde la posición actual.
- `absolute(int rowNumber)` Navega a la fila indicada.

Proceso de desarrollo

Liberando los recursos

Es importante liberar los recursos en memorio a través de método `close()` de todos los objetos utilizados: `Connection`, `Statement` y `ResultSet`

Proceso de desarrollo

Gestión de errores

Puede haber muchos errores potenciales

- Error en la conexión por ausencia o timeout.
- Error en la autenticación o en los permisos.
- Error en la sintaxis SQL
- Resultados vacios
- Valores NULL
- ...

Proceso de desarrollo

Gestión de errores

1. Comprobar siempre los valores devueltos
2. Proporcionar un código de manejo de excepciones coherente, **exception handlers**
3. Gestionar adecuadamente los errores debido a resultados vacíos o valores Nulos.
4. NUNCA enseñar al usuario errores de la base de datos

Proceso de desarrollo

Gestión de errores

1. ¿Cuáles son los errores que pueden surgir durante la carga del controlador? `ClassNotFoundException`
2. Manejo de los NULLs: NULL's de SQL $\neq$ NULL's de Java [2].
   - 2.1 Evitar usar métodos `getXXX( )` que devuelvan tipos de datos primitivos (usar `getObject()`).
   - 2.2 Usar clases de tipo Wrapper para tipos de datos primitivos.

```
aDouble = new Double (rs.getDouble(1));
System.out.println("before wasNull() aDouble = " + aDouble);
if (rs.wasNull()) {
    aDouble = null;
}
System.out.println("after wasNull() aDouble = " + aDouble);
```

   - 2.3 Usar tipos de datos primitivos.

```
ResultSet rs = stmt.executeQuery("SELECT id, first, last, age FROM Employees");
int id = rs.getInt(1);
if( rs.wasNull( ) ) {
     id = 0;
```

[2] Ejecuta NullTactics para ver lo que sucede

Proceso de desarrollo

Gestión de errores

3. Errores en el acceso a la BBDD: `SQLException`
   Debemos capturar o elevar este tipo de excepciones

```
try{
     // all statements including commit
}catch(SQLException e)
{
    // rollback or commit if rollback for a given savepoint
}
finally{
    // close connection or statements or resultsets
}
```

- int getErrorCode()
- String getSQLState()
- void setNextException(SQLException ex), SQLException getNextException() Iterator<Throwable> iterator()

Proceso de desarrollo

Warnings

- SQLWarning (heredan de SQLException) gestionan los warnings que se producen en el acceso a la base de datos (nivel de aislamiento no soportado, crear ResultSet no soportados . . .)
- Los Warnings NO detienen la ejecución del programa.
- `getWarnings()` obtiene warnings de los objetos Connection, ResultSet, y Statement.
- `clearWarrning()` borra los warnings del objeto anterior.

```
public void doSomething () {
  try {
    // Get warnings risen while connecting to the database
      SQLWarning connectionWarning = conn.getWarnings();
      while (connectionWarning != null) {
          String warningMessage = connectionWarning.getMessage();
          String warningSQLState = connectionWarning.getSQLState();
          int warningErrorCode = connectionWarning.getErrorCode();
          System.out.println("Connection warning : "
                   + warningErrorCode + " Message : " + warningMessage
                   + " SQL state " + warningSQLState);
          connectionWarning = connectionWarning.getNextWarning();
          }
```