everis
an NTT DATA Company

## JDBC

# BECA Java

El acceso a datos con JDBC

# JDBC

- Conceptos Básicos
- API JDBC
- Acceso a Base de datos
- Manipulación De Resultados
- Prepared Statements
- Callable Statements


# 01 Conceptos Básicos

# Acceso a datos con JDBC

## Funcionamiento de JDBC

Para realizar las operaciones comunes de recuperar o almacenar información en una base de datos la mayoría de lenguajes de programación utilizan un API o una librería clases auxiliar que permiten el envío de instrucciones SQL a la base de datos y la posterior manipulación de los resultados.

En el caso de Java, este API se llama JDBC, que son las siglas de **JAVA DATA BASE CONECTIVITY**. Se trata de un API independiente del tipo de base de datos . Esto significa que podríamos cambiar con el mismo código contra una base de datos SQL server u Oracle con tan sólo cambiar la cadena de conexión y el driver utilizado.

everis
an NTT DATA Company

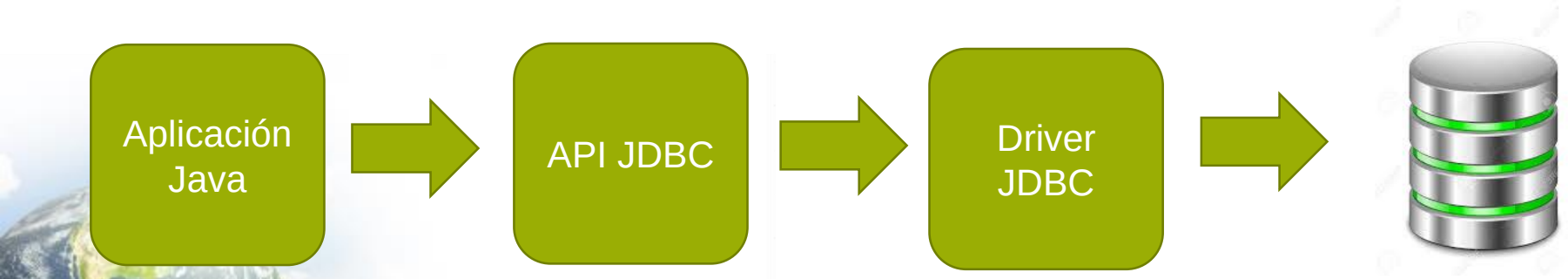
# Funcionamiento de JDBC

## El Driver JDBC

Esta compatibilidad es posible gracias a que el API no accede directamente a las BBDD, si no que en su lugar, accede a un software intermedio que traducirá las llamadas procedentes de la aplicación a instrucciones específicas de la Base de datos.

Podemos encontrar el driver específico de cada fabricante en su sitio web, los ejemplos más comunes:

- http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html
- https://dev.mysql.com/downloads/connector/j/



everis
an NTT DATA Company

02

# API JDBC

everis
an NTT DATA Company

# Programación con JDBC

## Las clases principales

Las principales clases e interfaces que forman el API JDBC se encuentran en el paquete java.sql. Esta sería una breve descripción de cada una de ellas:

- **DriverManager**: Esta clase es la encargada de establecer las conexiones con la base de datos a través del driver.
- **Connection**: Las conexiones con la base de datos implementa esta interfaz. Toda la transferencia de datos entre la aplicación y la BDD se realizará a través de un objeto Connection.
- **Statement**: Eta interfaz que proporciona los métodos necesarios para la ejecución de consultas de datos y recuperación de resultados.
- **PreparedStatrment**: al igual que la anterior se emplea para ejecutar operaciones sobre la base de datos utilizándose en este caso consultas pre compiladas.
- **Resulset**: Esta interfaz implementada por el objeto encargado de la manipulación de objetos procedentes de la BDD.
- **ResultSetMetadata**: Esta interfaz dispone de métodos para obtener información sobre la estructura de los campos recuperados en una consulta.

everis
an NTT DATA Company

# Programación con JDBC

## Interacción con la BDD

Anteriormente hemos comentado las características de las clase s e interfaces del API JDBC, a continuación vamos a analizar con detalle la manera en que éstas deben utilizarse para realizar las operaciones requeridas por la aplicación.

Normalmente, para realizar una operación que implique acceso a la BDD requerirá realizar lo siguiente:

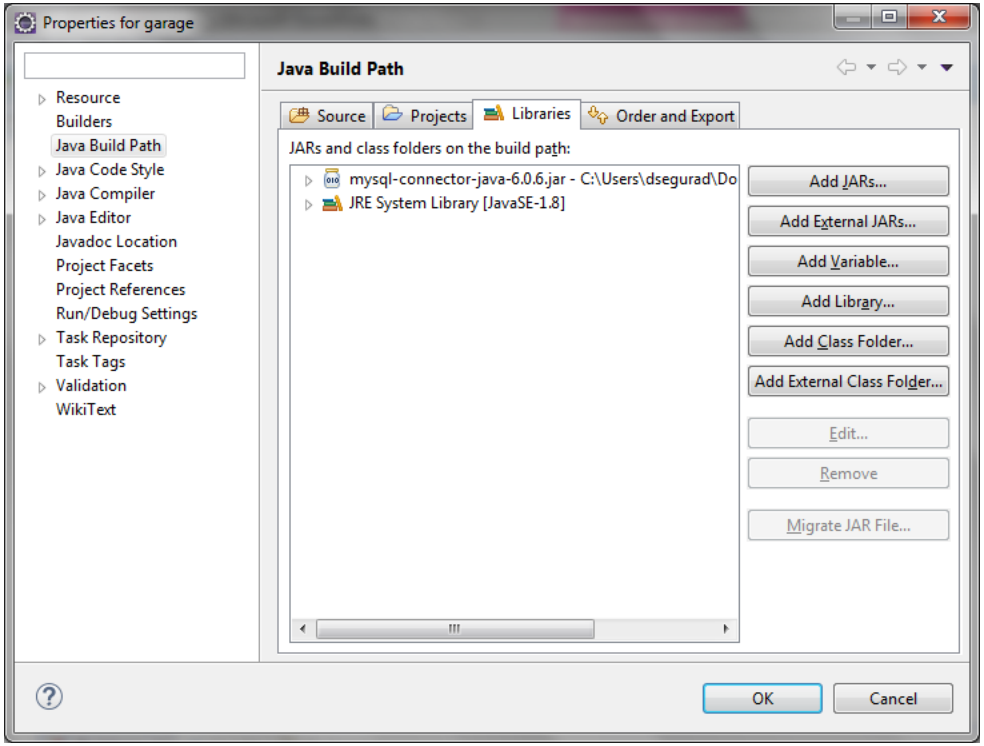
- Establecer una conexión con la BDD
- Ejecución de la consulta SQL o procedimiento
- Manipulación de los resultados (Si procede)
- Cierre de la conexión.

everis
an NTT DATA Company

# Programación con JDBC

## Añadir librería (jar)

Para acceder a las clases jdbc debemos añadir el jar correspondiente a nuestro proyecto. Para ello debemos agregar el jar correspondiente a tal como se aprecia en la captura siguiente:



everis
an NTT DATA Company

# Programación con JDBC

## Estableciendo conexión con la BDD

Dado que la conexión se establece a través del driver JDBC , lo primero que habrá que hacer será realizar la carga del mismo en memoria.

Esta operación se lleva a cabo utilizando el método estático forName() de la clase java.lang.Class, proporcionando como argumento en la llamada al mismo el nombre completo de la clase del driver.

En caso de conectar a una base de datos MySQL esta sería la sintaxis:

```
Class.forName(“com.mysql.jdbc.Driver)
```

# Programación con JDBC

## Obteniendo la conexión

Una vez cargado el driver se obtiene la conexión con la base de datos, para lo que tendremos que utilizar el primer método del driver JDBC. Se trata del método estático getConection() de la clase driver Manager. Este método necesita que se le pase como argumento la URL de la base de datos.

La URL de una base de datos depende del driver utilizado, siendo su sintaxis general:

Jdbc:subprotocolo:datos_adicionales

En el caso de específico de MySQL será necesario indicar la servidor de base de datos, el puerto utilizado y el nombre de la base de datos :

```
public Connection getConnection() throws SQLException{

    String url = "jdbc:mysql://localhost:3306/garage";

    try {
        //Carga del driver jdbc
        Class.forName("com.mysql.jdbc.Driver");
        //Conexion con la base de datos.

    } catch (ClassNotFoundException e) {

        System.out.println("Error al conectar con el driver JDBC");
    }
    Connection cn = DriverManager.getConnection(url);


    return cn;
}
```

everis
an NTT DATA Company

# Programación con JDBC

## Ejecución de consultas SQL

Una vez creado el objeto Connection , el resto de operaciones a realizar serán ya independientes del tipo de base de datos y del driver que estemos utilizando.

Utilizaremos el objeto Connection para crear un objeto Statement que nos permita enviar consultas SQL a la base de datos. Para ello deberemos invocar al método *createStatement()* proporcionado por la interfaz Connection:

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

Es importante tener en cuenta que todas estas operaciones pueden lanzar un SQLexception que será necesario capturar.

everis
an NTT DATA Company

# Programación con JDBC

## La interfaz Statement

Para la ejecución de la consulta la interfaz Statement dispone de tres métodos que utilizaremos según el tipo de sentencia SQL que se quiera enviar. Por ejemplo , si se trata de una consulta que no genera resultados (UPDATE, INSERT, DELETE), se empleará el método **execute**(), al cual habrá que proporcionar una cadena de texto con la sentencia SQL.

Además de ese método disponemos de **executeUpdate**() para ejecutar consultas SQL de acción. La diferencia con este último es que devuelve el numero de registros que se han visto afectados en la base de datos por la operación.

```
public void delete(Client client) throws IOException {


    String sql = "DELETE FROM CLIENTES WHERE NIF =' " + client.getDni() + "'";
    int total;

    try {

        Connection cn = this.getConnection();
        Statement st = cn.createStatement();
        total = st.executeUpdate(sql);
        System.out.println("Se han eliminado " + total + "registros");

        //Cerramos la conexión
        st.close();

    } catch (SQLException e) {
        System.out.println("Error al obtener la conexión");
    }
```

everis
an NTT DATA Company

# Programación con JDBC

## Ejercicio 1

Crearemos una nueva interfaz ClientDaoJDBC que contará con los métodos: create,read,delete,readAll, y getConnection.

Crearemos una nueva clase ClienteDaoJDBCImpl que implemente sus métodos . Esta implementación deberá ser capaz de al menos lo siguiente:

Insertar un nuevo cliente en la tabla de clientes de la base de datos Garage

- Informar al usuario de si el cliente se ha grabado correctamente.
- En caso de error deberemos capturar la SQLException y mostrar la información sobre el motivo del error.
- La tabla clientes se compondrá de los campos :
  - Dni (pk)
  - Nombre
  - Apellido
  - Fecha de nacimiento.

everis
an NTT DATA Company

# Programación con JDBC

## Ejercicio 2

Escribiremos el método delete que nos solicitará un dni  por pantalla.

- Informar al usuario de  si  el cliente se ha Eliminado correctamente.
- En caso de error deberemos capturar la SQLException y mostrar la información sobre el motivo del error.

# Programación con JDBC

## Manipulación de los resultados

Si la consulta que se quiere enviar a la base de datos es de tipo SELECT habrá que emplear el método executeQuery() de Statement en lugar de los comentados anteriormente.

Este método nos devuelve como resultado un objeto ResultSet que proporcionará acceso a los datos referidos por la instrucción SQL.

```
ResultSet  rs = st.executeQuery("SELECT * FROM CLIENTES");
```

En este caso el objeto ResultSet referenciado por la variable rs proporcionaría acceso a la aplicación a los datos de la tabla clientes.

El ResultSet proporcionará un acceso a los datos de modo solo lectura y solo avance. Esta limitación resulta más que suficiente en la mayoría de los casos, proporcionando además una buena velocidad de acceso un consumo limitado de recursos.

# Programación con JDBC

## Métodos de la interfaz ResultSet.

A través de los métodos ofrecidos por la interfaz ResulSet podremos desplazarnos por el conjunto de registros referidos por la instrucción SELECT y acceder al contenido de los campos . Estos son los métodos más comunes de la interfaz.

- **Next() :** Este método realiza un desplazamiento al siguiente registro del conjunto. Hay que tener en cuenta que inicialmente el objeto resulset se encuentra apuntando a la posición anterior al primer registro. Por lo que la primera vez que se llame pasará a apuntar a ese primer registro. El método next () devuelve un valor boolean que nos indica si está apuntando a un registro(true) o bien se ha desplazado más allá del último registro (false) por lo que resulta sencillo controlar el recorrido completo del conjunto.

everis
an NTT DATA Company

# Programación con JDBC

## Métodos de la interfaz ResultSet.

A través de los métodos ofrecidos por la interfaz ResulSet podremos desplazarnos por el conjunto de registros referidos por la instrucción SELECT  y acceder al contenido de los campos . Estos son los métodos más comunes de la interfaz.

- **getString(nombre del campo) :** Devuelve , para el registro apuntado actualmente, el valor del campo cuyo nombre se especifica. Como indica su nomobre, el dato es devuelto como un tipo String. Independientemente del tipo original con el que fue almacenado en la base de datos.

```
ResultSet rs  = st.executeQuery("SELECT * FROM CLIENTES");

while (rs.next()) {
    System.out.println(rs.getString("nombre"));
}
```

- **Get[tipodeDato](nombre del campo) :** Consisten en un conjunto de métodos que permiten recuperar el contenido del campo en el tipo primitivo Java correspondiente. Existe un método para cada tipo primitivo del lenguaje, siendo [tipodeDato] el nombre del tipo. Por ejemplo getInt() nos permitirá obtener el valor del campo como un entero mientras que el getBi¡gdecimal haría lo propio con el tipo de dato correspondiente.

## Programación con JDBC

### Métodos de la interfaz ResultSet.

La siguiente tabla muestra la equivalencia entre los tipos primitivos de Java y su correspondiente SQL

| SQL | Java |
| --- | --- |
| SMALLINT | short |
| INTEGER | int |
| INT | int |
| REAL | float |
| FLOAT | double |
| DOUBLE | double |
| BOOLEAN | boolean |

everis
an NTT DATA Company

# Programación con JDBC

## Cierre de la conexión

Los objetos Connection consumen una gran cantidad de recursos del sistema , por lo que se hace necesario realizar el cierre de las conexiones con la base de datos una vez realizadas todas las operaciones pertinentes sobre la misma.

Esta operación se hace invocando el método **close()** que se ha utilizado para el cierre de los objetos Statement y ResultSet

Si no se han cerrado explícitamente los objeto Statement y ResultSet, el cierre del objeto connection provocará también el cierre automático de estos objetos.

everis
an NTT DATA Company

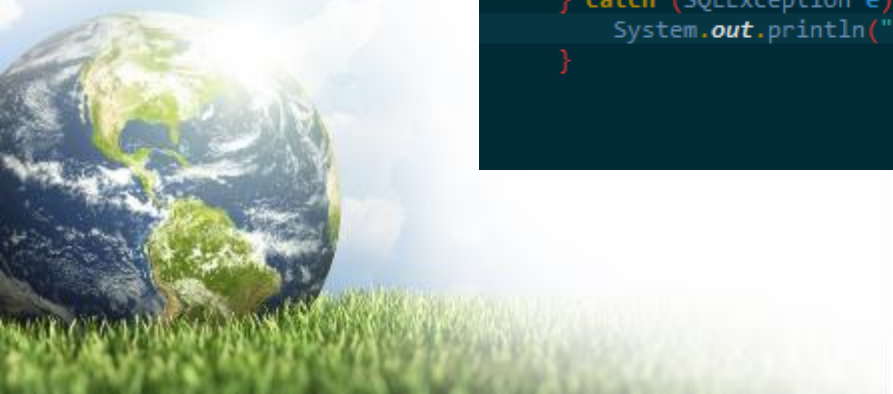
# Programación con JDBC

## La excepción SQLException

Como hemos visto anteriormente, la llamada a los distintos métodos del API JDBC puede provocar una excepción del tipo SQL. Esta clase dispone de tres métodos que pueden aportar información valiosa sobre los errores producidos.

- **getMessage()** : Devuelve una cadena de caracteres con la información asociada a la excepción producida.
- **getErrorCode():** Devuelve un código de error específico del fabricante, por lo que habrá que consultar la información sobre el driver utilizado para conoce los posibles códigos de error que puede devolver el método y su significado.
- **getSqlState():** Devuelve una cadena de caracteres de cinco cifras con un código de error estandarizado que indica el estado de la sentencia SQLal producirse la excepción.

```
} catch (SQLException e) {
    System.out.println("Error:" + e.getMessage());
}
```



# Programación con JDBC

## Ejercicio 3

Escribiremos el método read  que implementará el correspondiente método read de la interfaz.

Los requerimientos para este método serán los siguientes:

- Solicitará  por consola el  dni del cliente  cuyos datos queremos consultar.
- En caso de que la consulta devuelva  resultado, mostraremos por pantalla todos los datos del cliente solicitado.
- En caso de que no se encuentre el cliente , mostraremos un mensaje indicando que no hemos encontrado resultados.
- En caso de error deberemos capturar la SQLException y mostrar la información sobre el motivo del error.

# 04 Prepared Statements

everis
an NTT DATA Company

# Programación con JDBC

## Las Consultas preparadas

Supongamos que tenemos que ejecutar repetidas veces una misma instrucción SQL sobe la base de datos, variando con cada ejecución únicamente ciertos criterios de la condición , como por ejemplo, una consulta que nos permita actualizar el precio de una determinada reparación variando sólo la marca o el modelo del coche.

En estos casos en lugar de construir la sentencia cada vez que se quiere realizar su ejecución resulta más eficiente pre compilar la sentencia y ejecutarla después las veces que sea necesario.

Dado que no todos los datos de la sentencia son conocidos durante la pre compilación , pueden utilizarse parámetros que serán posteriormente sustituidos por valores concretos durante la ejecución.

Para ello usaremos la interfaz **PreparedStatement** en lugar de la Statement.

everis
an NTT DATA Company

# Programación con JDBC

## Precompilación de sentencias con PreparedStatement

Un obejeto preparedStatement se crea a partir del método preparedStatemnt(consulta) de la interfaz Connection a la que habrá que suministrar como argumento la cadena de texto con la consulta SQL correspondiente.

Como se indicó anteriormente se pueden definir parámetros dentro de una consulta preparada. Dichos parámetros se representan por el símbolo de interrogación “?” . El siguiente ejemplo

```
@Override
public void create(Client client) throws IOException {

try {

    String sql = " INSERT INTO CLIENTES "
            + " VALUES ( ?,?,?,?)" ;
```

everis
an NTT DATA Company

# Programación con JDBC

## Asignación de parámetros.

La llamada a PrepareStatement() genera una consulta pre compilada, lista para su ejecución, pero antes de ello, se deberán asignar valores a los parámetros definidos en la consulta. Esta operación se lleva a cabo mediante un conjunto de métodos proporcionados por la interfaz .

```
setTipodato ( posicion_parametro, valor);
```

Existe un método de estas características para cada uno de los tipos primitivos de Java, además de String , Object y Date. Por ejemplo para asignar un valor de tipo entero a un parámetro, utilizaríamos **setInt(),** mientras que si es de tipo String el método llamado sería **setString().**

everis
an NTT DATA Company

# Programación con JDBC

## Asignación de parámetros.

El primer argumento de estos métodos representa el orden que ocupa el parámetro dentro de la instrucción, mientras que el segundo constituye el valor a asignar.

Es importante destacar que a diferencia de la consultas creadas explícitamente, donde los valores de los campos no numéricos no deben ir encerrados entre comillas, el simbolo “?” se escribirá tal cual , sin entrecomillado , independientemente del tipo de dato utilizado. Para la ejecución de la consulta se utilizarán los mismos métodos definidos en la interfaz statement. Execute(), executeUpdate(), executeQuery, según el caso.

```
String sql = " INSERT INTO CLIENTES "
        + " VALUES ( ?,?,?,?)" ;

    Connection cn = this.getConnection();
    PreparedStatement pst = cn.prepareStatement(sql);

    pst.setString(1, "52776563");
    pst.setInt(2, 345);
    pst.setDate(3, java.sql.Date.valueOf("10/01/1973"));
```

everis
an NTT DATA Company

# Programación con JDBC

## Ejercicio 4

Escribiremos Una nueva clase que extienda de ClientDaoJDBImpl e implemente los métodos de la interfaz ClientDaoJDBC. Esta nueva clase cumplirá los mismos requerimientos de la interfaz anterior pero hará uso de preparedStatement en lugar de la interfaz Statement.

# 03 Procedimientos Almacenados

everis
an NTT DATA Company

# Programación con JDBC

## Procedimientos Almacenados

En ocasiones los desarrolladores de aplicaciones deciden almacenar parte del código SQL en la propia base de datos en forma de procedimientos almacenados, para posteriormente ejecutarlos desde la aplicación en lugar de generar las instrucciones de la misma.

Este sistema tiene como ventaja el aumento del rendimiento y la simplificación del código de la aplicación. Por el contrario se tiene una mayor dependencia del tipo de base de datos, dificultando la portabilidad.

everis
an NTT DATA Company

# Programación con JDBC

## La interfaz CallableStatement

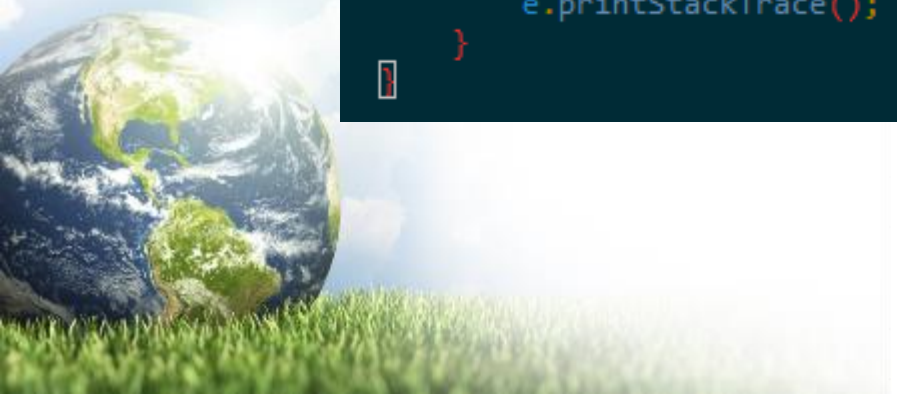
La ejecución de un procedimiento almacenado desde una aplicación JDBC se realiza mediante un objeto CallableStatement el cual puede obtenerse invocando al método **prepareCall()** de la interfaz Connection.

Este método requiere como parámetro el nombre del procedimiento que se quiere ejecutar.

La interfaz Callable Statement() es una subinterfaz de preparedStatement, siendo su uso bastante similar ella.

El siguiente ejemplo obtiene un objeto CallableStatement asociado a un procedimiento almacenado llamado “baja cliente” Que realiza la eliminación de un cliente de la base de datos.

```
try {
    java.sql.CallableStatement cbs = cn.prepareCall("bajaCliente");
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
```



everis
an NTT DATA Company

# Programación con JDBC

## Asignación de parámetros

Dado que el procedimiento puede requerir la asignación de parámetros para su ejecución , la interfaz CallableStatement proporciona una serie de métodos para asignar valores a los mismos. Estos métodos tienen el formato:

```
setTipodato ( nombreParámetro, valor);
```

Donde tipoDato representa el nombre de cad uno de los tipos primitivos de Java , además de String, Object y Date. Así mismo el nombreParámetro es una cadena de caracteres que indica el nombre del parámetro y “valor” el valor que se quiere asignar al mismo.