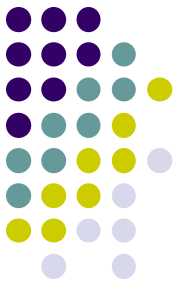
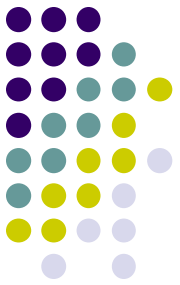


# JDBC

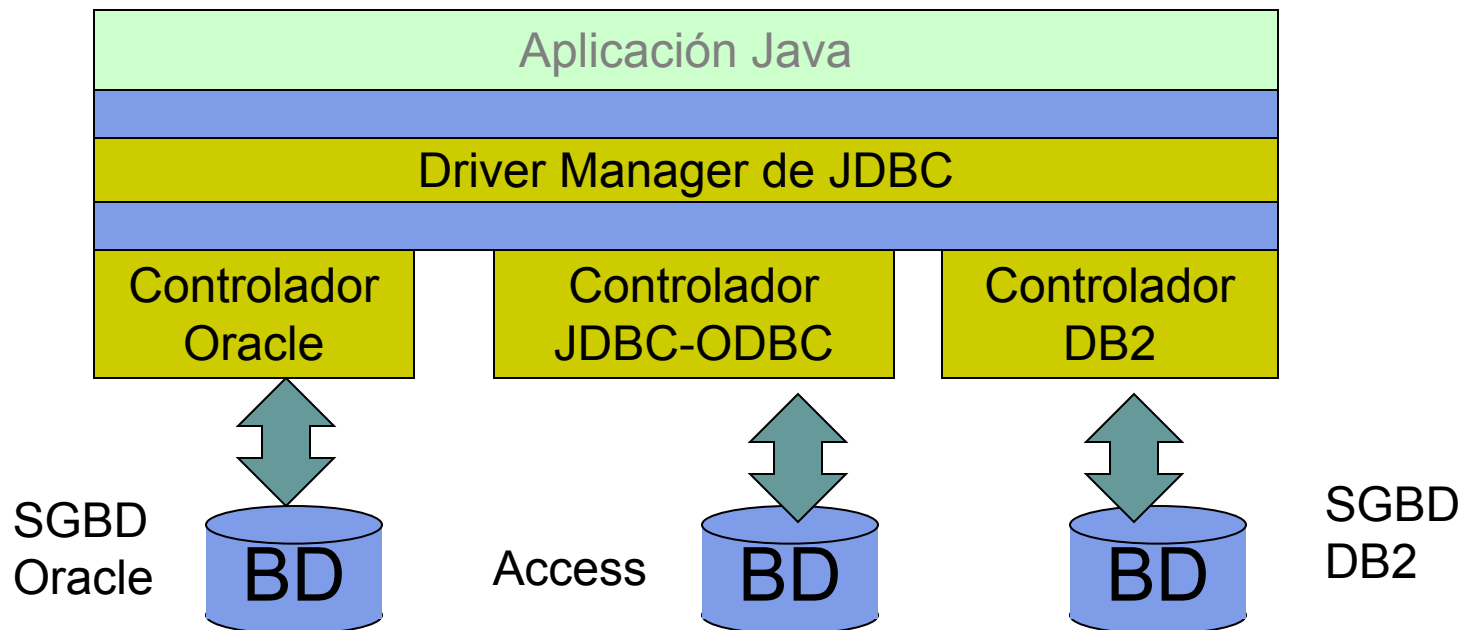


- Es un interfaz orientado a objetos de Java para SQL.
- Se utiliza para enviar sentencias SQL a un sistema gestor de BD (DBMS).
- Con JDBC tenemos que continuar escribiendo las sentencias SQL.
- No añade ni quita potencia al SQL.

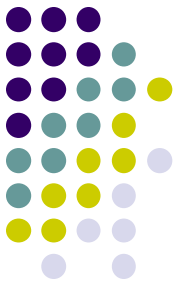


# Arquitectura JDBC

- La filosofía de JDBC es proporcionar transparencia al desarrollador frente al gestor de BD.
- JDBC utiliza un *Gestor de Controladores* que hace de interfaz con el controlador específico de la BD.

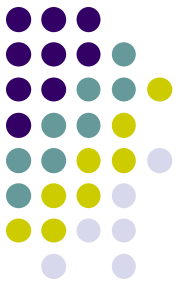


# MySQL

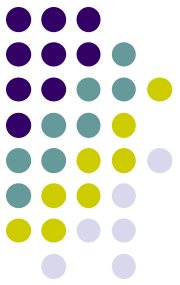


- Instalación del driver JDBC
  - Descomprimos el fichero `mysql-connector-java-5.0.6.zip`
  - Añadimos la librería `mysql-connector-java-5.0.6-bin.jar`
    - Si compilamos desde línea de comandos, añadimos el fichero a la variable de sistema `CLASSPATH`
    - Si usamos Eclipse, Project > Properties > Java Build Path > Libraries > Add External JARs...
  - Driver: `com.mysql.jdbc.Driver`
  - URL: `jdbc:mysql://localhost:3306/sample`

# JDBC

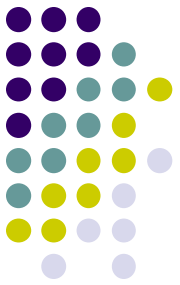


- Introducción a las bases de datos
- Bases de datos en Java. JDBC
  - Introducción a JDBC
  - Diseño de una aplicación con BD
  - Conexiones a la base de datos
  - Sentencias SQL
  - Transacciones
  - Uso de **ResultSet**



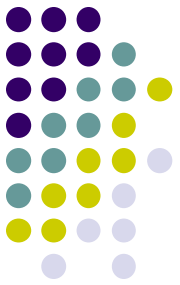
# Introducción a JDBC

- *Java DataBase Connectivity*
- Es la API (librería) estándar de acceso a base de datos desde Java
- Está incluida en Java SE (*Standard Edition*)
- En Java SE 6 se incluye JDBC 4.0, pero actualmente la mayoría de bases de datos soportan JDBC 3.0
- Más información
  - <http://java.sun.com/javase/technologies/database>
  - <http://java.sun.com/docs/books/tutorial/jdbc/>



# Introducción a JDBC

- Para conectarse a una base de datos concreta, es necesario su driver JDBC
- El driver es un fichero JAR que se añade a la aplicación como cualquier otra librería (no necesita instalación adicional)
- La mayoría de las bases de datos incorporan un driver JDBC
- ODBC (*Open DataBase Connectivity*) es un estándar de acceso a base de datos desarrollado por Microsoft. Sun ha desarrollado un driver que hace de puente entre JDBC y ODBC aunque no suele usarse.



# Introducción a JDBC

- Los pasos para que una aplicación se comuniquen con una base de datos son:
  1. Cargar el driver necesario para comprender el protocolo que usa la base de datos concreta
  2. Establecer una conexión con la base de datos, normalmente a través de red
  3. Enviar consultas SQL y procesar el resultado
  4. Liberar los recursos al terminar
  5. Manejar los errores que se puedan producir



```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

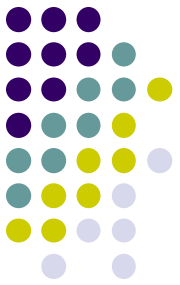
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");

        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

Bambi	3.0
Batman	4.0





# Ejercicio 1

- Implementar el ejemplo anterior
- Comprobar su funcionamiento
- En las siguientes transparencias se explicará en detalle el significado de cada una sus partes



```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");

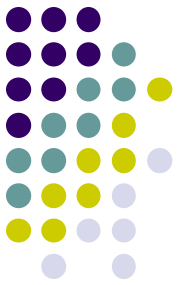
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");

        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

Carga del driver

# Introducción a JDBC

## Carga del driver



- Antes de poder conectarse a la base de datos es necesario cargar el driver JDBC
- Sólo hay que hacerlo una única vez al comienzo de la aplicación

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

- El nombre del driver debe venir especificado en la documentación de la base de datos
- Se puede elevar la excepción **ClassNotFoundException** si hay un error en el nombre del driver o si el fichero .jar no está correctamente en el **CLASSPATH** o en el proyecto



```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

Establecer una  
conexión

```
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");
```

```
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");
```

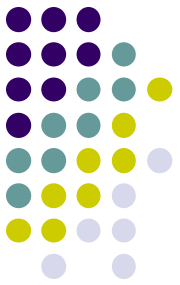
```
        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
```

```
        rs.close();
        stmt.close();
        conn.close();
```

```
    }
```

```
}
```

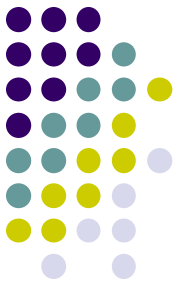
# Establecer una conexión



- Las bases de datos actúan como servidores y las aplicaciones como clientes que se comunican a través de la red
- Un objeto **Connection** representa una conexión física entre el cliente y el servidor
- Para crear una conexión se usa la clase **DriverManager**
- Se especifica la URL, el nombre y la contraseña

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://localhost:3306/sample","root","pass");
```

# Establecer una conexión



- El formato de la URL debe especificarse en el manual de la base de datos
- Ejemplo de MySQL

```
jdbc:mysql://<host>:<puerto>/<esquema>
```

```
jdbc:mysql://localhost:3306/sample
```

- El nombre de usuario y la contraseña dependen también de la base de datos



```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

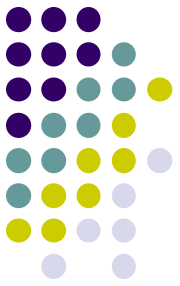
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");

        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

**Ejecutar una  
sentencia SQL**

# Ejecutar una sentencia SQL



- Una vez que tienes una conexión puedes ejecutar sentencias SQL
- Primero se crea el objeto **Statement** desde la conexión
- Posteriormente se ejecuta la consulta y su resultado se devuelve como un **ResultSet**

```
Statement stmt = conn.createStatement();  
ResultSet rs = stmt.executeQuery(  
    "SELECT titulo, precio FROM Libros WHERE precio > 2");
```





```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

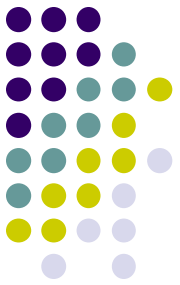
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");

        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

**Acceso al conjunto de resultados**

# Acceso al conjunto de resultados

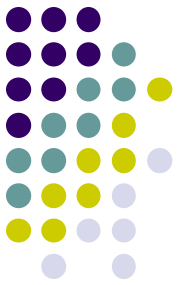


- El `ResultSet` es el objeto que representa el resultado
- No carga toda la información en memoria
- Internamente tiene un cursor que apunta a una fila concreta del resultado en la base de datos
- Hay que posicionar el cursor en cada fila y obtener la información de la misma

```
while (rs.next()) {  
    String name = rs.getString("titulo");  
    float price = rs.getFloat("precio");  
    System.out.println(name + "\t" + price);  
}
```

# Acceso al conjunto de resultados

## Posicionamiento del cursor

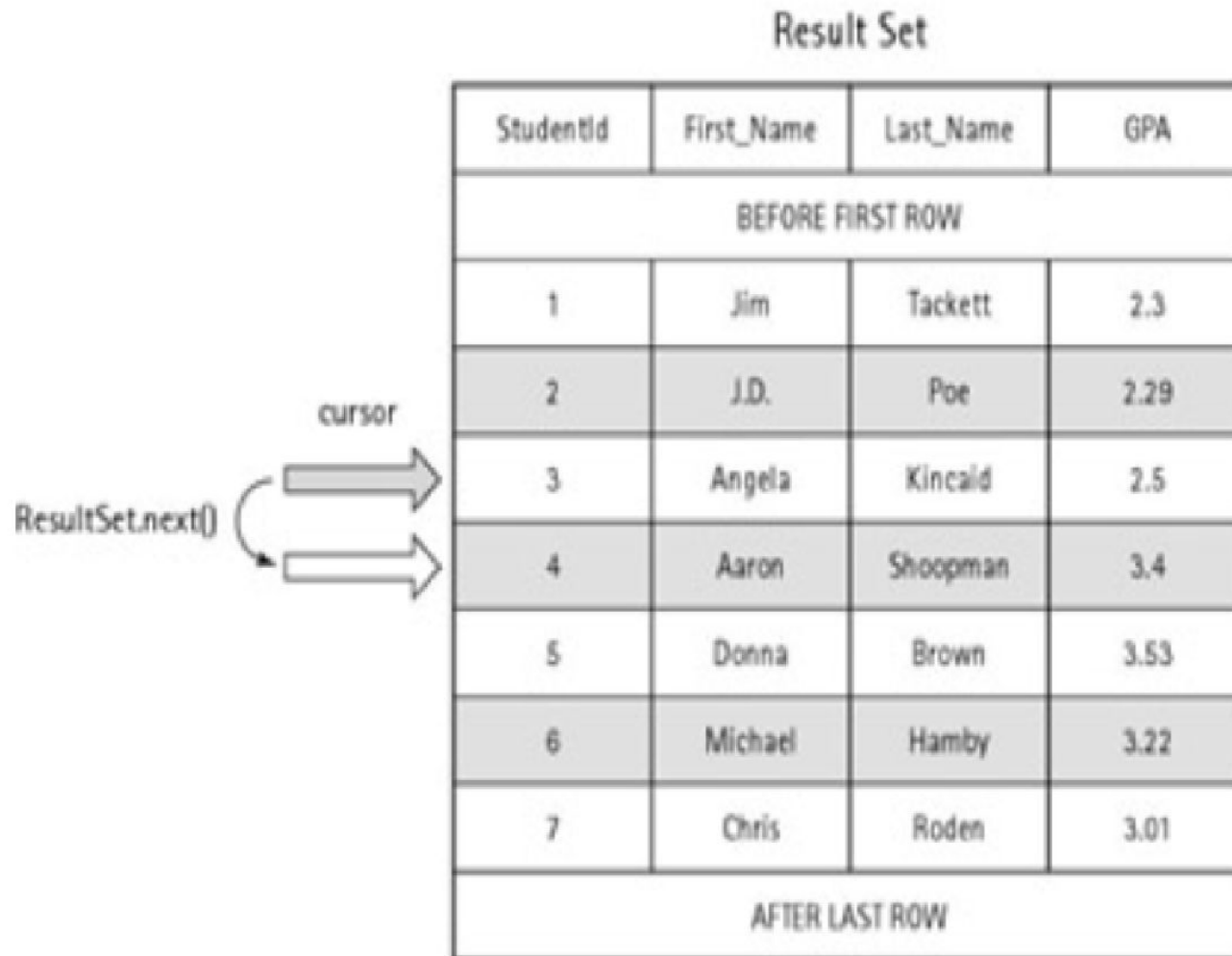
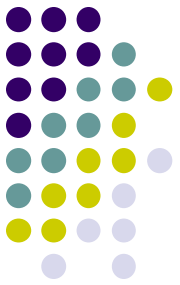


- El cursor puede estar en una fila concreta
- También puede estar en dos filas especiales
  - Antes de la primera fila (*Before the First Row*, BFR)
  - Después de la última fila (*After the Last Row*, ALR)
- Inicialmente el **ResultSet** está en BFR
- **next()** mueve el cursor hacia delante
  - Devuelve **true** si se encuentra en una fila concreta y **false** si alcanza el ALR

```
while (rs.next()) {  
    String name = rs.getString("titulo");  
    float price = rs.getFloat("precio");  
    System.out.println(name + "\t" + price);  
}
```

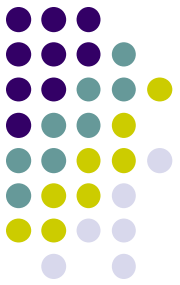
# Acceso al conjunto de resultados

## Posicionamiento del cursor



# Acceso al conjunto de resultados

## Obtención de los datos de la fila



- Cuando el `ResultSet` se encuentra en una fila concreta se pueden usar los métodos de acceso a las columnas
  - `String getString(String columnLabel)`
  - `String getString(int columnIndex)`
  - `int getInt(String columnLabel)`
  - `int getInt(int columnIndex)`
  - ... (existen dos métodos por cada tipo)

Los índices  
empiezan en  
1 (no en 0)

```
while (rs.next()) {  
    String name = rs.getString("titulo");  
    float price = rs.getFloat("precio");  
    System.out.println(name + "\t" + price);  
}
```



```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

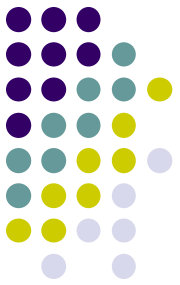
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");

        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

**Librerar Recursos**

# Liberar recursos



- Cuando se termina de usar una **Connection**, un **Statement** o un **ResultSet** es necesario liberar los recursos que necesitan
- Puesto que la información de un **ResultSet** no se carga en memoria, existen conexiones de red abiertas
- Métodos **close()**:
  - **ResultSet.close()** – Libera los recursos del **ResultSet**. Se cierran automáticamente al cerrar el **Statement** que lo creó o al reejecutar el **Statement**.
  - **Statement.close()** – Libera los recursos del **Statement**.
  - **Connection.close()** – Finaliza la conexión con la base de datos



## Manejar los errores

```
import java.sql.*;
public class HolaMundoBaseDatos {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {

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

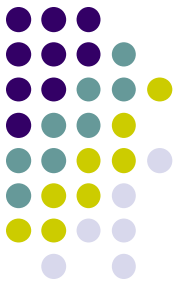
        Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/sample","root","pass");

        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(
            "SELECT titulo, precio FROM Libros WHERE precio > 2");

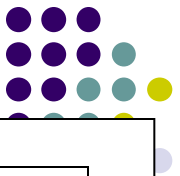
        while (rs.next()) {
            String name = rs.getString("titulo");
            float price = rs.getFloat("precio");
            System.out.println(name + "\t" + price);
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}
```



# Manejar los errores



- Hay que gestionar los errores apropiadamente
- Se pueden producir excepciones **ClassNotFoundException** si no se encuentra el driver
- Se pueden producir excepciones **SQLException** al interactuar con la base de datos
  - SQL mal formado
  - Conexión de red rota
  - Problemas de integridad al insertar datos (claves duplicadas)




```
import java.sql.*;
public class HolaMundoGestionErrores {
    public static void main(String[] args) {

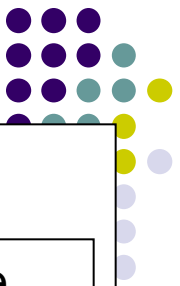
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("El driver no se encuentra");
            System.exit(-1);
        }

        Connection conn = null;
        try {
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sample", "root", "pass");

            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(
                "SELECT titulo, precio FROM Libros WHERE precio > 2");
```

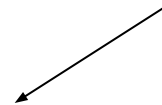
Gestión de  
errores en la  
localización del  
driver





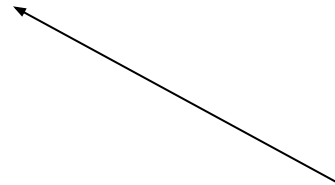
```
while (rs.next()) {  
    String name = rs.getString("titulo");  
    float price = rs.getFloat("precio");  
    System.out.println(name + "\t" + price);  
}  
rs.close();  
stmt.close();
```

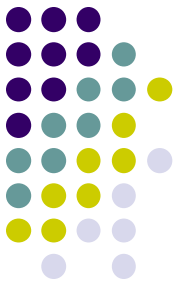
Gestión de  
errores en el  
envío de  
consultas



```
} catch (SQLException e) {  
    System.err.println("Error en la base de datos: "+  
        e.getMessage());  
    e.printStackTrace();  
}  
finally {  
    if(conn != null){  
        try {  
            conn.close();  
        } catch (SQLException e) {  
            System.err.println("Error al cerrar la conexión: "+  
                e.getMessage());  
        }  
    }  
}
```

Gestión de  
errores al cerrar  
la conexión

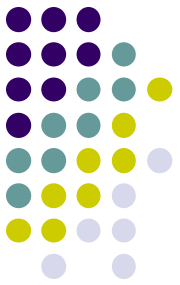




## Ejercicio 2

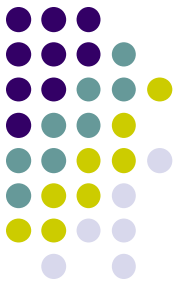
- Implementar el ejercicio anterior
- Comprobar la gestión de errores provocando errores
  - Cambia el nombre del driver
  - Cambia el formato de la URL
  - Modifica la sentencia SQL

# JDBC



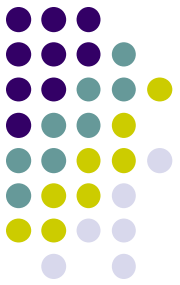
- Introducción a las bases de datos
- Bases de datos en Java. JDBC
  - Introducción a JDBC
  - Diseño de una aplicación con BD
  - Conexiones a la base de datos
  - Sentencias SQL
  - Transacciones
  - Uso de **ResultSet**

# JDBC



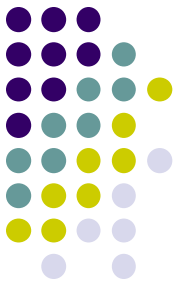
- Introducción a las bases de datos
- Bases de datos en Java. JDBC
  - Introducción a JDBC
  - Diseño de una aplicación con BD
  - Conexiones a la base de datos
  - Sentencias SQL
  - Transacciones
  - Uso de **ResultSet**

# Conexiones a la base de datos



- Cada objeto **Connection** representa una conexión física con la base de datos
- Se pueden especificar más propiedades además del usuario y la password al crear una conexión
- Estas propiedades se pueden especificar:
  - Codificadas en la URL (ver detalles de la base de datos)
  - Usando métodos **getConnection(...)** sobrecargados de la clase **DriverManager**

# Conexiones a la base de datos



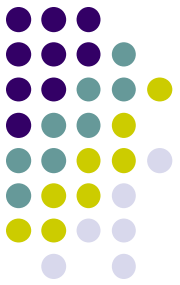
```
String url = "jdbc:mysql://localhost:3306/sample";  
String name = "root";  
String password = "pass" ;  
Connection c = DriverManager.getConnection(url, user,  
password);
```

```
String url =  
  
"jdbc:mysql://localhost:3306/sample?user=root&password=pass";  
Connection c = DriverManager.getConnection(url);
```

```
String url = "jdbc:mysql://localhost:3306/sample";  
Properties prop = new Properties();  
prop.setProperty("user", "root");  
prop.setProperty("password", "pass");  
Connection c = DriverManager.getConnection(url, prop);
```

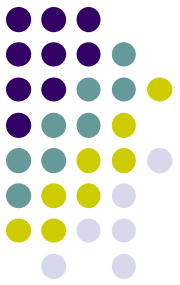


# JDBC

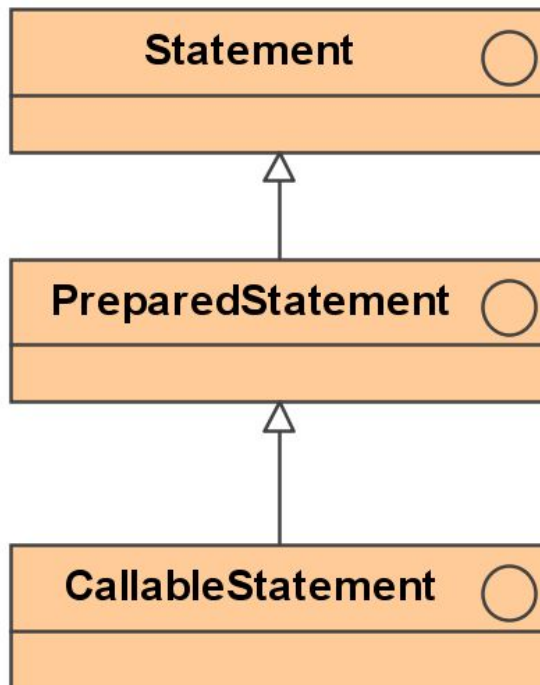


- Introducción a las bases de datos
- Bases de datos en Java. JDBC
  - Introducción a JDBC
  - Diseño de una aplicación con BD
  - Conexiones a la base de datos
  - Sentencias SQL
  - Transacciones
  - Uso de **ResultSet**

# Sentencias SQL



- Con JDBC se pueden usar diferentes tipos de **Statement**



SQL estático en tiempo de ejecución, no acepta parámetros

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

Para ejecutar la misma sentencia muchas veces (la “prepara”). Acepta parámetros

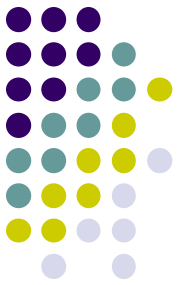
```
PreparedStatement ps =  
    conn.prepareStatement(...);
```

Llamadas a procedimientos almacenados

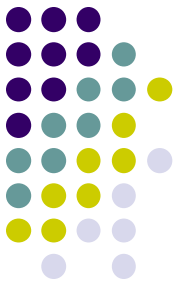
```
CallableStatement s =  
    conn.prepareCall(...);
```

# Sentencias SQL

## Uso de Statement



- Tiene diferentes métodos para ejecutar una sentencia
  - **executeQuery ( . . . )**
    - Se usa para sentencias SELECT. Devuelve un ResultSet
  - **executeUpdate (...)**
    - Se usa para sentencias INSERT, UPDATE, DELETE o sentencias DDL. Devuelve el número de filas afectadas por la sentencia
  - **execute (...)**
    - Método genérico de ejecución de consultas. Puede devolver uno o más ResultSet y uno o más contadores de filas afectadas.

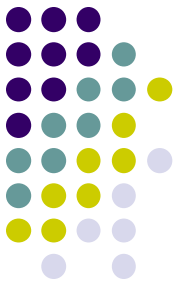


# Ejercicio 4

- Incorpora a la librería
  - Inserción de libros (con sus autores)

```
INSERT INTO Libros VALUES (1,'Bambi',3)
INSERT INTO Autores VALUES (1,'Pedro','Húngaro')
INSERT INTO relacionlibroautor VALUES (1,1)
```

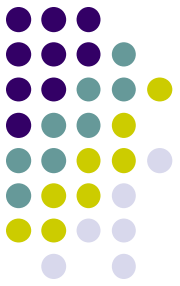
# Ejercicio 4



- Se puede seguir el siguiente esquema para la inserción:
  - Pedir datos libro (también el id)
  - Preguntar número de autores
  - Por cada autor
    - Preguntar si es nuevo
    - Si es nuevo
      - Pedir datos autor (también el id)
      - Insertar autor en base datos (Insertamos datos en la tabla Autores)
      - Guardar Autor en la lista de autores del libro
    - si no
      - Pedir código del autor
      - Cargar el autor
      - Guardar Autor en la lista de autores del libro
  - Insertar nuevo libro (Insertamos datos en las tablas Libros y RelacionLibroAutor)

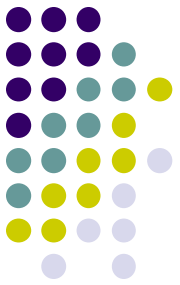
# Sentencias SQL

## Uso de PreparedStatement



- Los PreparedStatement se utilizan:
  - Cuando se requieren parámetros
  - Cuando se ejecuta muchas veces la misma sentencia
    - La sentencia se prepara al crear el objeto
    - Puede llamarse varias veces a los métodos **execute**

```
PreparedStatement ps = conn.  
    preparedStatement("INSERT INTO Libros VALUES (?, ?, ?)");  
  
ps.setInt(1, 23);  
ps.setString(2, "Bambi");  
ps.setInt(3, 45);  
  
ps.executeUpdate();
```



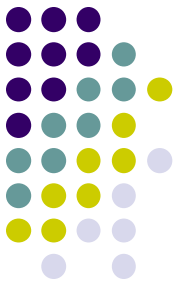
# Ejercicio 5

- Incorpora a la librería
  - Borrado de libros (implementado con **PreparedStatement**)

```
DELETE FROM relacionlibroautor WHERE idLibro = 1  
DELETE FROM libros WHERE idLibro = 1
```

# Sentencias SQL

## Uso de CallableStatement

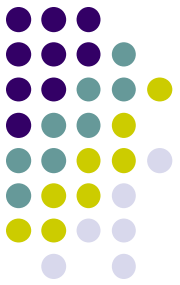


- Permite hacer llamadas a los procedimientos almacenados de la base de datos
- Permite parámetros de entrada IN (como el **PreparedStatement**), parámetros de entrada-salida INOUT y parámetros de salida OUT

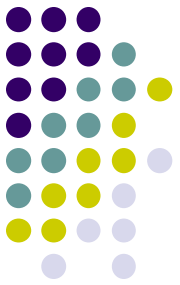
```
CallableStatement cstmt =  
    conn.prepareCall ("{call getEmpName (?,?)}");  
  
cstmt.setInt(1,111111111);  
cstmt.registerOutParameter(2,java.sql.Types.VARCHAR);  
  
cstmt.execute();  
  
String empName = cstmt.getString(2);
```



# JDBC



- Introducción a las bases de datos
- Bases de datos en Java. JDBC
  - Introducción a JDBC
  - Diseño de una aplicación con BD
  - Conexiones a la base de datos
  - Sentencias SQL
  - Transacciones
  - Uso de **ResultSet**



# Transacciones

- Las transacciones tratan un conjunto de sentencias como una única sentencia, de forma que si una falla, todo lo anterior se deshace

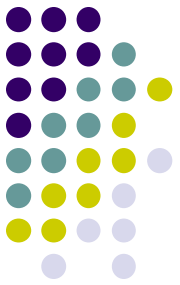
```
try {  
    conn.setAutoCommit(false);  
    Statement statement =  
    conn.createStatement();  
    statement.executeUpdate("DELETE ...");  
    statement.executeUpdate("DELETE ...");  
    conn.commit();  
    conn.setAutoCommit(true);  
    statement.close();  
} catch (SQLException e) {  
    try {  
        conn.rollback();  
    } catch (SQLException e1) {  
        System.err.println("Error");  
    }  
    System.err.println("Error");  
}
```

Por defecto se hace commit por cada sentencia. Hay que desactivarlo

Cuando se han ejecutado todas las sentencias, se hace "commit"

Si algo falla, se hace "rollback"

Se vuelve a poner el autocommit, para el resto de la aplicación

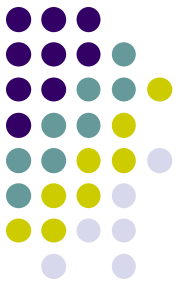


# Ejercicio 6

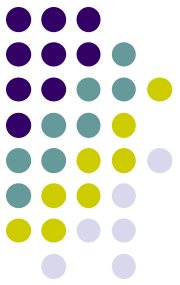
- Incorpora a la librería
  - Borrado de libros con transacciones
  - De esta forma o se asegura de que se borran todas las filas de todas las tablas y no se deja la base de datos inconsistente

```
DELETE FROM relacionlibroautor WHERE idLibro = 1  
DELETE FROM libros WHERE idLibro = 1
```

# JDBC



- Introducción a las bases de datos
- Bases de datos en Java. JDBC
  - Introducción a JDBC
  - Diseño de una aplicación con BD
  - Conexiones a la base de datos
  - Sentencias SQL
  - Transacciones
  - Uso de `ResultSet`

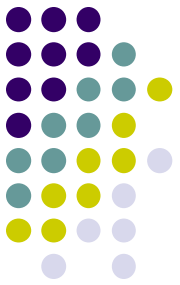


# Uso de ResultSet

- El **ResultSet** es el objeto que representa el resultado de una consulta
- No carga toda la información en memoria
- Se pueden usar para actualizar, borrar e insertar nuevas filas

# Uso de ResultSet

## Características

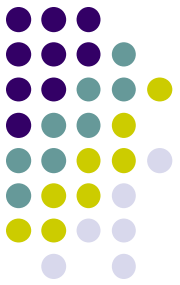


- Al crear un `Statement`, un `PreparedStatement` o un `CallableStatement`, se pueden configurar aspectos del `ResultSet` que devolverá al ejecutar la consulta

```
createStatement(  
    int resultSetType, int resultSetConcurrency);  
  
prepareStatement(String SQL,  
    int resultSetType, int resultSetConcurrency);  
  
prepareCall(String sql,  
    int resultSetType, int resultSetConcurrency);
```

# Uso de ResultSet

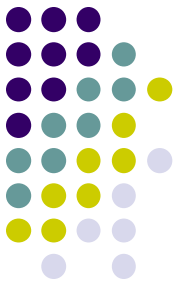
## Características



- Características del **ResultSet**
  - resultSetType
    - ResultSet.TYPE\_FORWARD\_ONLY – Sólo movimiento hacia delante (por defecto)
    - ResultSet.TYPE\_SCROLL\_INSENSITIVE – Puede hacer cualquier movimiento pero no refleja los cambios en la base de datos
    - ResultSet.TYPE\_SCROLL\_SENSITIVE – Puede hacer cualquier movimiento y además refleja los cambios en la base de datos
  - resultSetConcurrency
    - ResultSet.CONCUR\_READ\_ONLY – Sólo lectura (por defecto)
    - ResultSet.CONCUR\_UPDATABLE - Actualizable

# Uso de ResultSet

## Características



- Actualización de datos

```
rs.updateString("campo", "valor");  
rs.updateInt(1, 3);  
rs.updateRow();
```

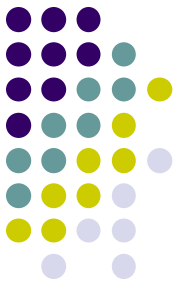
- Inserción de datos

```
rs.moveToInsertRow();  
rs.updateString(1, "AINSWORTH");  
rs.updateInt(2, 35);  
rs.updateBoolean(3, true);  
rs.insertRow();  
  
rs.moveToCurrentRow();
```

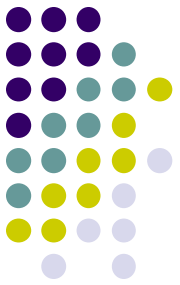
Mueve el cursor a la posición anterior al movimiento a inserción



# Ejercicio 7



- Incorpora a la librería
  - Reducción del precio de todos los libros a la mitad de precio
  - Utilizando un **ResultSet** actualizable



## Uso de ResultSet

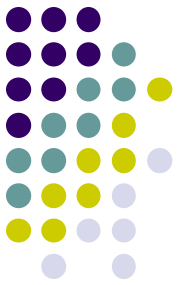
# Posicionamiento del cursor

- El cursor puede estar en una fila concreta
- También puede estar en dos filas especiales
  - Antes de la primera fila (*Before the First Row*, BFR)
  - Después de la última fila (*After the Last Row*, ALR)
- Inicialmente el **ResultSet** está en BFR
- **next()** mueve el cursor hacia delante
  - Devuelve **true** si se encuentra en una fila concreta y **false** si alcanza el ALR

```
while (rs.next()) {  
    String name = rs.getString("titulo");  
    float price = rs.getFloat("precio");  
    System.out.println(name + "\t" + price);  
}
```

# Uso de ResultSet

## Posicionamiento del cursor

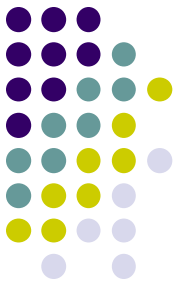


Result Set

StudentId	First_Name	Last_Name	GPA
BEFORE FIRST ROW			
1	Jim	Tackett	2.3
2	J.D.	Poe	2.29
3	Angela	Kincald	2.5
4	Aaron	Shoopman	3.4
5	Donna	Brown	3.53
6	Michael	Hamby	3.22
7	Chris	Roden	3.01
AFTER LAST ROW			

Diagram illustrating the cursor movement:

The cursor is positioned at the first row (StudentId 1). The `ResultSet.next()` method is shown moving the cursor to the next row (StudentId 2).



# Uso de ResultSet

- Métodos que permiten un movimiento por el ResultSet
  - next() – Siguiente fila
  - previous() – Fila anterior
  - beforeFirst() – Antes de la primera
  - afterLast() – Después de la última
  - first() – Primera fila
  - last() – Última fila
  - absolute() – Movimiento a una fila concreta
  - relative() – Saltar ciertas filas hacia delante