JDBC

Una mini-introducción

Introducción

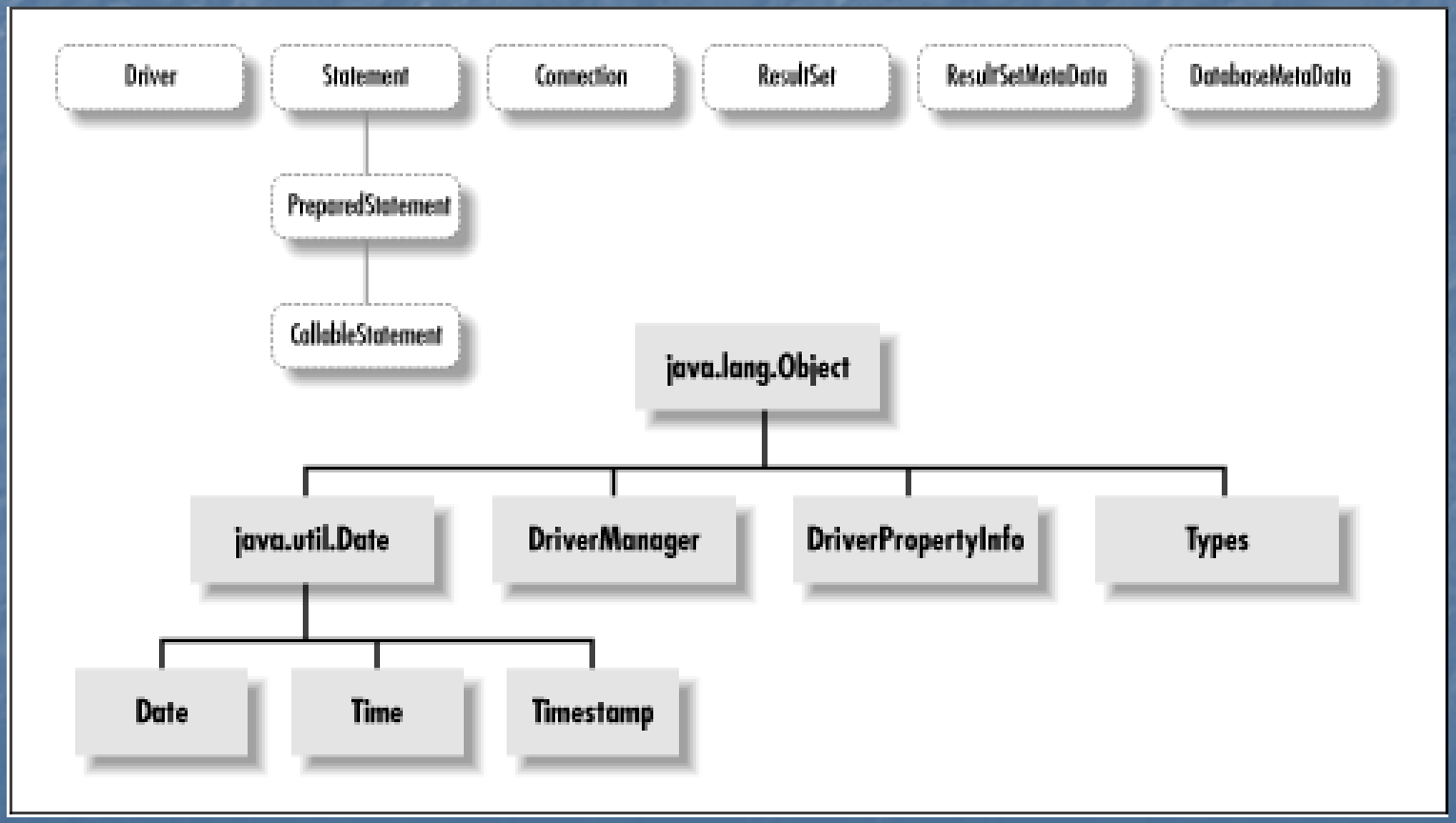
- JDBC (Java DataBase Connectivity)
  - Protocolo para utilizar bases de datos relacionales desde Java
  - Se basa en la utilización de drivers que implementan un API predefinido
- Una vez seleccionado el driver el resto del código es independiente del SGBD

Paquetes Java para JDBC

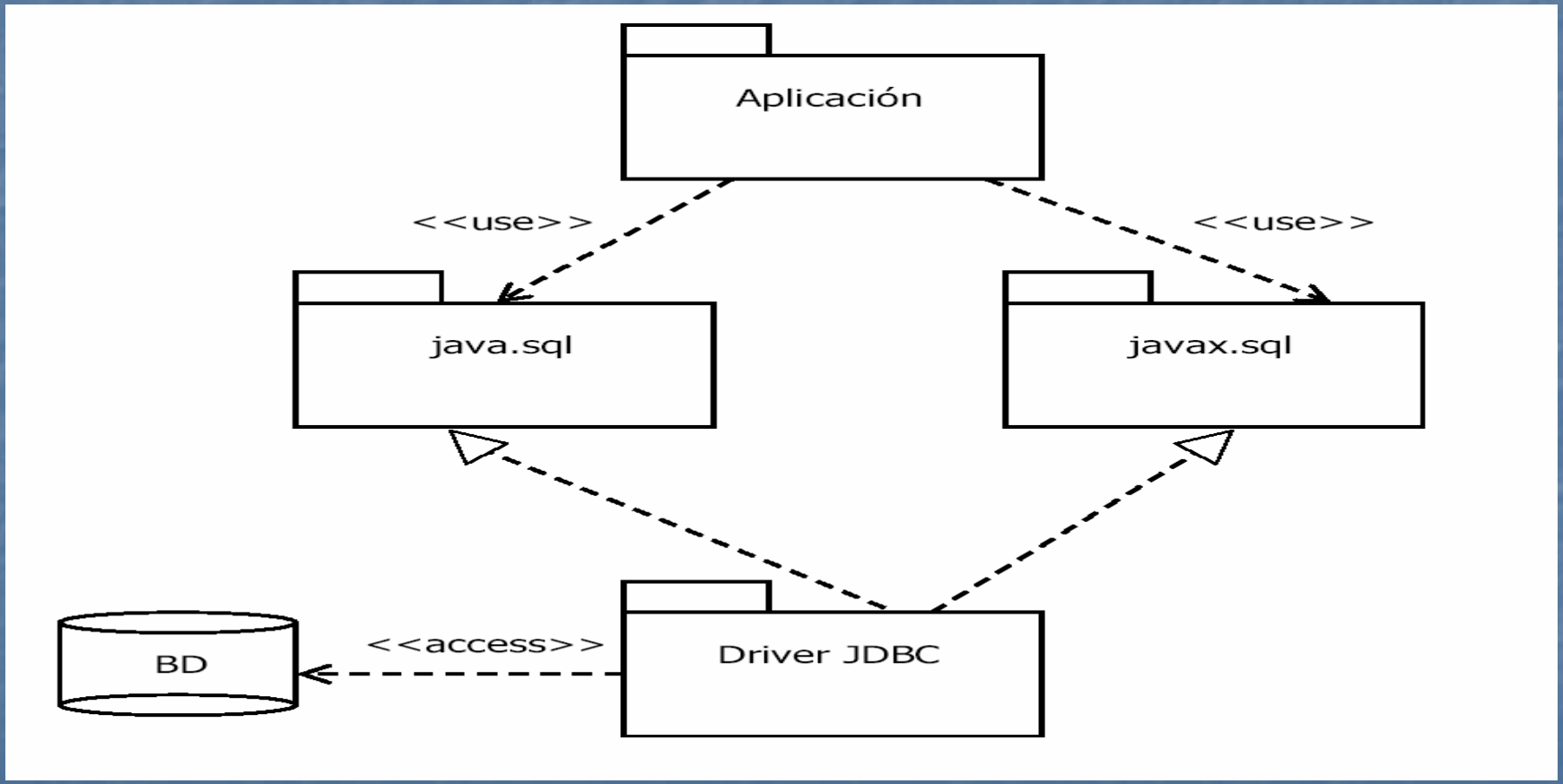
- 2 paquetes
  - java.sql
  - javax.sql (más avanzado)
- La mayor parte de las aplicaciones sólo requieren java.sql

java.sql

- Clases e interfaces:



Estructura JDBC



Para comunicar Java con un SGBD

- Hay que conocer:
  - Fichero físico que contiene el driver (jar/zip) (debe estar incluido en el CLASSPATH de la aplicación)
  - Nombre de la clase driver de Java (opcional)
  - URL de conexión
- Estos valores dependen de cada SGBD, e incluso del driver concreto

Algunos driver JDBC

* MySql
  - Clase Driver : com.mysql.jdbc.Driver
  - URL de Conexión: jdbc:mysql://<host>/<database>
  - Fichero .jar/.zip: mysql-connector-java-5.0.4-bin.jar
* DB2
  - Clase Driver : com.ibm.db2.jdbc.app.DB2Driver
  - URL de Conexión:: jdbc:db2:<database>
  - Fichero .jar/.zip: db2java.zip
* Sybase
  - Clase Driver : com.sybase.jdbc2.jdbc.SybDriver
  - URL de Conexión: jdbc:sybase:Tds:<host>:<port>/<database>
  - Fichero .jar/.zip: jconn2.jar
* Oracle
  - Clase Driver : oracle.jdbc.driver.OracleDriver
  - URL de Conexión: jdbc:oracle:thin:@ <host>:<port>:<sid>
  - Fichero .jar/.zip: classes12.zip
* SQLServer
  - Clase Driver : com.microsoft.jdbc.sqlserver.SQLServerDriver
  - URL de Conexión: jdbc:microsoft:sqlserver://localhost:1433
  - Fichero .jar/.zip: mssqlserver.jar, msbase.jar, msutil.jar
* PostgreSQL
  - Clase Driver: org.postgresql.Driver
  - URL de Conexión: jdbc:postgresql://<server>:<port>/<database>
  - Fichero .jar/.zip: postgresql.jar

Programas Java con JDBC

- Fases:
  1. Establecer la conexión con la BBDD (clase Connection)
  2. Crear una sentencia SQL (clase Statement o PreparedStatement)
  3. Lanzar la sentencia
  4. Tratar el resultado (clase ResultSet)

Fase 1: Establecer la conexión (ejemplo)

```
Connection connection = null
Statement statement = null;
ResultSet resultSet = null;
try
    {
        String driverClassName = "com.mysql.jdbc.Driver";
        String driverUrl = "jdbc:mysql://localhost/barcos";
        String user = "bertoldo";
        String password = "gominolas";
        Class.forName(driverClassName);
        connection = DriverManager.getConnection(driverUrl,
                                                 user,  password);
        .....
```

Fase 1.1

- Comprobamos la existencia del driver:

```
...
try
    {
        String driverClassName = "com.mysql.jdbc.Driver";
        String driverUrl = "jdbc:mysql://localhost/barcos";
        String user = "bertoldo";
        String password = "gominolas";
        Class.forName(driverClassName);
        connection = DriverManager.getConnection(driverUrl,
                                                 user,  password);
      .....
```

- Si se produce una excepción debe comprobarse que el fichero físico existe y está en el classpath

Fase 1.2

- Nombre de la conexión URL

```
...
try
    {
      ...
          String driverUrl = "jdbc:mysql://localhost/barcos";
.....
```

- La forma general de la conexión para mysql es:

  jdbc:mysql://<host>/<database>

Fase 1.3

- Abrimos la conexión:

```
Connection connection = null
Statement statement = null;
ResultSet resultSet = null;
try
    {
        String driverClassName = "com.mysql.jdbc.Driver";
        String driverUrl = "jdbc:mysql://localhost/barcos";
        String user = "bertoldo";
        String password = "gominolas";
        Class.forName(driverClassName);
        connection = DriverManager.getConnection(driverUrl,
                                                 user,  password);
      .....
```

- Normalmente los nombres de usuario y su password los introduce el usuario

Fases 2 y 3

- La sentencia depende del "dialecto" SQL del SGBD:

```
…
Statement statement = null;
ResultSet resultSet = null;
try
    {
            …
                    statement = connection.createStatement();

                    String query = "SELECT * FROM  Batallas;";
                    resultSet = statement.executeQuery(query);
```

- La sentencia puede ser cualquier válida en SQL del SGBD (Insert, Delete, Create Table, etc..)

Fase 4

- Tratar el resultado:

```
…
ResultSet resultSet = null;
try
    {
            …

     while (resultSet.next()) {
            // una forma de obtener una columna: por posición
            String nombre = resultSet.getString(1);
            // otra forma de obtener una columna: por su nombre
            Date fecha = resultSet.getDate("fecha");
            System.out.println("Nombre: " + nombre+ "| Fecha: "+fecha);
    }
```

Errores

- Se pueden producir errores por:
  - Fallo en la conexión
  - No existe la base de datos o no se tienen permisos sobre ella
  - Error de sintaxis en la sentencia SQL
  - Operación no permitida
- Se producirá una excepción en el programa

Excepciones

```
            …
try
    {

            … // las 4 fases descritas anteriormente

    } catch (ClassNotFoundException e) {
                    System.out.println("No se encuentra el driver");
    } catch (SQLException E) {
                    System.out.println("Excepcion SQL: " + E.getMessage());
                    System.out.println("Estado SQL:     " + E.getSQLState());
                    System.out.println("Código del Error:  " + E.getErrorCode());
            }
```

Liberar Recursos

- Aunque no se llame a Connection.close, cuando la conexión sea eliminada por el garbage collector, el método finalize de la clase que implementa Connection, invocará al método close
  - Cuando se cierra una conexión, cierra todos sus Statements asociados
  - Cuando se cierra un Statement, cierra todos sus ResultSets asociados
- Pero:
  - En una aplicación multi-thread que solicita muchas conexiones por minuto (ej.: una aplicación Internet)
  - Puede haber bugs en algunos drivers, de manera que no cierren los Statements asociados a una conexión o los **ResultSets** asociados a un Statement

Es imprescindible cerrar las conexiones tan pronto como se pueda

Liberando recursos

```
…
try
    {
            … // las 4 fases descritas anteriormente

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

PreparedStatement

- Cada vez que se envía una query a la BD, ésta:
  - La analiza sintácticamente
  - Construye un plan para ejecutarla
- Si tenemos un bucle en el que repetidamente se lanza la misma query con distintos parámetros → ineficiencia usando Statement
- En este tipo de situaciones, es mejor usar PreparedStatement

Ejemplo PreparedStatement

- Ejemplo:

```
// establecer la conexión
 …

String [] nombreBatallas={…..};
Date [] fechaBatallas={…..};  // de la misma long. que el anterior
String query = "INSERT INTO Batallas (nombre,fecha) VALUES (?,?);";
preparedStatement = connection.prepareStatement(queryString);
/* insertamos las batallas en la BD */
for (int i=0; i<nombreBatallas.length; i++) {
          // rellenamos el parámetro 1  y 2
         preparedStatement.setString(1, nombreBatallas[i]);
         preparedStatement.setDate(2, balances[i]);
         // ejecutar la consulta
         int filas = preparedStatement.executeUpdate();
         if (filas != 1) { throw new SQLException("Problemas insertando "+ nombreBatallas[i]);
 }
…
```

Ejemplo PreparedStatement (II)

- En el ejemplo anterior:
  - Sólo se hace el análisis sintáctico una vez
  - Se hace un único plan de ejecución
- Mejor eficiencia con respecto a Statement

Lo que no hemos contado

- Hemos visto java.sql pero no javax.sql
  - Utiliza el tipo DataSource para cargar el driver → más configurable
- Uso del "pool de conexiones" para múltiples accesos a la misma base de datos
- Usar transacciones no atómicas:
  - Cambiar a connection.setAutoCommit(false);
  - Acabar con connection.commit() si todo va bien o
  - Con connection.rollback() si hay algún problema
  - Problemas de concurrencia
- Scrollable ResulSets, sentencias batch, etc., etc.