Práctica 4 - Java, JDBC y bases de datos

OBJETIVO

El objetivo de esta práctica es desarrollar la **capa de acceso a los datos** de una **aplicación de bases** de datos para la gestión de conciertos del Palau de la Música (base de datos desarrollada en la práctica 3). La aplicación se escribirá en el lenguaje de programación Java y hará uso de la API JDBC para la implementación de las operaciones de acceso a la base de datos.

INTRODUCCIÓN

Una **aplicación de base de datos** consiste en una base de datos junto con la aplicación o conjunto de aplicaciones coordinadas que se ejecutan utilizando los servicios de un sistema de gestión de bases de datos. Una aplicación de base de datos almacena sus datos en una base de datos y utiliza un sistema de gestión de base de datos para recuperar, actualizar, proteger y mantener la integridad de los datos.

De acuerdo con esto, la aplicación de base de datos debe implementar los mecanismos necesarios para acceder y manipular los datos contenidos en la base de datos. Este mecanismo se conoce como la **capa de acceso a los datos**, abreviadamente **DAO** (del inglés ***D**ata **A**ccess **O**bjects*). La mayor parte de esta interacción se reduce a los procesos de crear, consultar, actualizar y borrar la información. El conjunto de estas operaciones se conoce con el acrónimo **CRUD**, cuyo origen está en las siglas de su nombre en inglés (***C**reate, **R**ead, **U**pdate* y ***D**elete*).

Para la implementación de las operaciones CRUD se utilizará la API JDBC. Esta API proporciona una forma simple y consistente de trabajar con bases de datos relacionales. La idea que subyace en el diseño de JDBC es que el desarrollador interactúe con la base de datos siempre del mismo modo, independientemente de producto de gestión de bases de datos con el que se esté trabajando.

ESPECIFICACION DEL PROBLEMA

Para la realización de esta práctica, partiremos de una aplicación ya funcional que trabaja sobre la base de datos utilizada en las prácticas anteriores y que puede ser útil como modelo del trabajo a realizar. La aplicación consta de dos paneles funcionales: gestión de la tabla MI_MASCOTAS (como se muestra en la Ilustración 1) y un explorador de SQL que permite realizar consultas genéricas sobre la base de datos.

En la Ilustración 2 se muestra la estructura de clases del proyecto de ejemplo. La clase `es.uv.bd.Animalia` es la clase que implementa el método `main` desde donde se instancian la vista y el controlador. El controlador `es.uv.bd.controller.AnimaliaController` en este caso se encarga sólo de la gestión del menú de la aplicación. En el paquete `es.uv.bd.view` se implementan todas las clases relacionadas con la interface gráfica proporcionada por la aplicación, así como los controladores internos utilizados por cada uno de los paneles. Esta aplicación carece de modelo formal, ya que ese papel lo realizan las clases DAO que comentaremos a continuación.

Ilustración 1. Panel de gestión de la tabla de mascotas

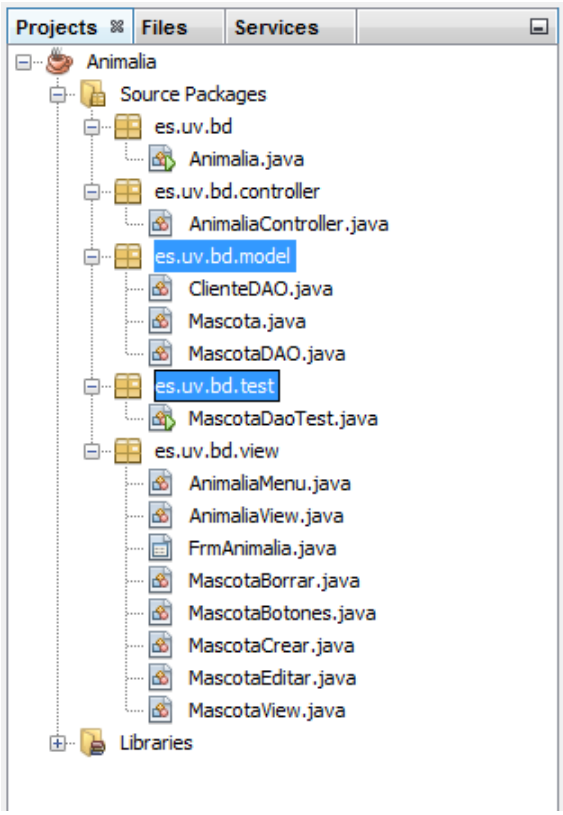

Ilustración 2. Estructura de paquetes y clases del proyecto de ejemplo

Las clases más interesante desde el punto de vista de la realización de esta práctica son las clases contenidas en el paquete `es.uv.bd.model`: `MascotaDAO` (que implementa mediante JDBC todas operaciones de acceso a la tabla de mascotas) y `Mascota` (clase sencilla que permite transferir información desde y hacia el DAO).

La clase `MascotaDAO` implementa cinco métodos: los cuatro métodos CRUD (`crearMascota`, `leerMascota`, `actualizarMascota` y `borrarMascota`) y un método de consulta `getTablaMascotas` que proporciona un `DefaultTableModel` que es utilizado en la vista `MascotaView` para mostrar la información de todas las mascotas. En la Ilustración 3 se asigna a la variable `READ` la sentencia SQL de búsqueda por clave primaria y en la Ilustración 4 se muestra un fragmento del método `leerMascota`.

```
1  private static final String READ =
2       "SELECT idmascota, idcliente, nombremascota, tipoanimal, " +
3       "       TO_CHAR(fechanac,'DD-MM-YYYY') AS fecha" +
4       "  FROM mi_mascotas " +
5       " WHERE idmascota = ?";
```

Ilustración 3. Sentencia SQL de búsqueda de una mascota por su identificador de mascota

```
1  public Mascota leerMascota(int idMascota) throws ClassNotFoundException,
2    InstantiationException, IllegalAccessException, SQLException, ParseException {
3      Mascota mascota = new Mascota();
4      SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
5
6      Class.forName(DRIVER).newInstance();
7      Connection oracleConn = DriverManager.getConnection(DBURL,USERNAME,PASSWORD);
8
9      PreparedStatement read = oracleConn.prepareStatement(READ);
10     read.setInt(1, idMascota);
11     ResultSet rs = read.executeQuery();
12
13     if (rs.next()) {
14        mascota.setIdMascota(rs.getInt("idmascota"));
15        mascota.setIdCliente(rs.getInt("idcliente"));
16        mascota.setNombreMascota(rs.getString("nombremascota"));
17        mascota.setTipoAnimal(rs.getInt("tipoanimal"));
18        Date fechaNacimiento = sdf.parse(rs.getString("fecha"));
19        mascota.setFechaNacimiento(fechaNacimiento);
20     }
21     return mascota;
22 }
```

Ilustración 4. Fragmento de código del método leerMascota de la clase MascotaDAO

En todos los métodos del DAO, las operaciones realizadas implican (ver Ilustración 4):

1. Obtener un gestor de Driver de JDBC (línea 6) utilizando el nombre de la clase. A partir de éste se instancia una conexión (línea 7) a la base de datos utilizando los datos de la conexión. Los parámetros de conexión se definen en las constantes `DRIVER`, `DBURL`, `USERNAME` y `PASSWORD`.

   Importante: Para ejecutar el proyecto es necesario personalizar las variables `USERNAME` y `PASSWORD` (ahora no tienen valor). Modifique estas variables en las clases `MascotaDAO` y `ClienteDAO` y los parámetros equivalentes en la clase `FrmAnimalia` (línea 157). Utilice los datos de conexión correspondientes a su usuario y palabra de paso.

2. Construir un `PreparedStatement` (línea 9) a partir de la sentencia SQL (almacenada en la variable `READ`) y asignar valores a los parámetros. En nuestro caso sólo hay un parámetro y se asigna en la línea 10.
3. Ejecutar la sentencia (línea 11).
4. En el caso se consultas, es necesario recorrer el `ResultSet` resultante para procesar los valores devueltos en la consulta. El bloque de código 13-20 construye a partir del `ResultSet` un objeto de la clase `Mascota` que devuelve en la línea 21.

Por último, la clase `es.uv.bd.test.MascotaDaoTest` es una clase de utilidad cuya misión es comprobar el correcto funcionamiento de los DAO realizando una serie de operaciones controladas y mostrando el resultado a través de la consola del sistema. Para ejecutar esta clase, basta con seleccionar la clase en el árbol de proyecto y teclear `Mayúscula+F6` o elegir esta opción en el menú contextual.

TRABAJO A REALIZAR

Construcción de la capa DAO. En este apartado se desarrollará la capa de acceso a las tablas **Salas** y **Conciertos** (base de datos desarrollada en la práctica 3). Recordad que la tabla Conciertos es débil con respecto a la tabla Salas.

El trabajo consiste en el desarrollo de las clases DAO `SalasDAO` y `ConciertosDAO` y los objetos de transferencia de datos `Salas` y `Conciertos`. También será necesario desarrollar las clases de prueba de los DAO creados (`SalasDaoTest` y `ConciertosDaoTest`).

DOCUMENTACIÓN A ENTREGAR

Proyecto NetBeans (empaquetado en formato .zip), con el código desarrollado. Se valorará:

1. El nivel del desarrollo alcanzado.
2. La calidad del código escrito, muy especialmente el de las clases mencionadas en el trabajo a realizar.
3. La funcionalidad del programa.

FECHA DE ENTREGA

La **FECHA MÁXIMA DE ENTREGA DE LA PRÁCTICA** se establecerá en la tarea correspondiente en el aula virtual.