MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Bases de datos orientadas a objetos I

18

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Introducción a las bases de datos orientadas a objetos 4

/ 3. Características de las bases de datos orientadas a objetos 4

/ 4. Caso práctico 1: “Comprendiendo las bases de datos orientadas a objetos.” 5

/ 5. Gestores de bases de datos orientadas a objetos 5

/ 6. Conectores Java-DB4O 6

/ 7. Agregar bibliotecas al proyecto 7

/ 8. Conectar, desconectar e inserción de datos 8

/ 9. Consultas de datos 9

9.1. Consultas de datos con condiciones 9

/ 10. Actualización de datos 10

/ 11. Caso práctico 2: “¿Se puede seguir usando SQL?” 11

/ 12. Borrado de datos 11

/ 13. Aplicando la POO 12

/ 14. Resumen y resolución del caso práctico de la unidad 13

/ 15. Bibliografía 13

OBJETIVOS

Conocer los diferentes modelos de bases de datos.

Instalar un gestor de bases de datos orientadas a objetos.

Realizar programas con acceso a bases de datos orientadas a objetos.

Insertar datos en una base de datos orientada a objetos.

Recuperar información de una base de datos orientada a objetos.

Borrar y modificar datos de una base de datos orientada a objetos.

/ 1. Introducción y contextualización práctica

En esta unidad vamos a estudiar un nuevo tipo de base de datos, las bases de datos orientadas a objetos y sus características.

Vamos a ver qué gestores de bases de datos soportan este tipo de base de datos y cómo podemos utilizarlas dentro de nuestros proyectos.

Además, vamos a aprender a realizar las operaciones típicas de toda base de datos, como insertar, actualizar, borrar y consultar datos con diferentes criterios de búsqueda.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Fig. 1. Analogía de base de datos.

Audio Intro. "Base de datos orientada a objetos VS Base de datos relacional"
http://bit.ly/35vPx81

/ 2. Introducción a las bases de datos orientadas a objetos

Hasta el momento hemos estado utilizando un único tipo de bases de datos, **las bases de datos relacionales, en las que la información se almacena en forma de tabla, compuestas por filas que almacenan la información de los datos.** En este tipo de bases de datos existen las reglas de integridad, que rigen las relaciones entre tabla para que haya el mínimo contenido repetido.

A parte de las bases de datos relacionales **existen más tipos**, como, por ejemplo, entre otras:

- Bases de datos jerárquicas.
- Base de datos de red.
- Bases de datos transaccionales.
- Bases de datos multidimensionales.
- Bases de datos orientadas a objetos.
- Bases de datos deductivas.
- Bases de datos documentales.

Cada una de estas bases de datos son muy diferentes entre sí y tienen uso en áreas muy específicas.

Nosotros nos vamos a centrar en esta unidad en las bases de datos orientadas a objetos.

En este tipo de bases de datos la información no se va a representar mediante tablas, sino **mediante objetos como los que se utilizan en la programación dirigida a objetos.**

Los gestores de bases de datos que soportan este tipo de bases de datos se denominan **Sistemas Gestores de Bases de Datos Orientados a Objetos, SGBDOO** por sus siglas en español o **ODBMS** por sus siglas en inglés.

Este tipo de bases de datos fueron diseñadas para poder trabajar codo con codo con los lenguajes de programación orientados a objetos y tener así una mayor integración entre programa y base de datos.

Fig. 2. Integrando una base de datos.

/ 3. Características de las bases de datos orientadas a objetos

Las bases de datos orientadas a objetos tienen las siguientes características:

- Soportan un **modelo de orientación a objetos puro**, utilizando las mismas definiciones de datos que utiliza el lenguaje de programación orientado a objetos que las utilice.
- Soportan los pilares de la programación orientada a objetos, como son el **encapsulamiento, la ocultación de información, la herencia y el polimorfismo**.
- Los objetos que estén dentro de la base de datos tendrán **un identificador único** generado por el propio sistema, haciendo uso de él para las tareas de borrado y de actualización. Este identificador no tiene nada que ver con la clave primaria de las tablas de las bases de datos relacionales.

- Se pueden definir **jerarquías de tipos** y hacer uso de la **herencia para los tipos almacenados**.
- Los objetos que se almacenen en la base de datos pueden contener objetos como **variables** dentro de ellos, es decir, se puede usar la **relación de composición de las clases**.
- El acceso a los datos se hará de una forma **navegacional**, es decir, este acceso se hará navegando por la estructura que haya creada por los objetos.
- Existe la **gestión de versiones**, es decir, podrá haber varias versiones de un mismo objeto dentro de la base de datos.
- Estas bases de datos pueden ser **portables**. Si se utiliza un gestor de bases de datos orientadas a objetos que no necesite instalación, la base de datos se reducirá a un fichero, el cual podremos mover de un sitio a otro sin problema. Esto es un arma de doble filo, ya que si el fichero se corrompe o se borra se perderá toda la base de datos.
- Al almacenar **objetos**, cuando recuperemos uno o varios de ellos, éstos tendrán **toda la información dentro**, es decir, una vez recuperados serán totalmente funcionales.

Fig. 3. Recuperando datos.

/ 4. Caso práctico 1: "Comprendiendo las bases de datos orientadas a objetos."

Planteamiento: Pilar y José están observando que existen más tipos de bases de datos a parte de las relacionales. Una de estas nuevas bases de datos son las orientadas a objetos, lo cual deja a nuestros amigos un poco descolocados. "Con las bases de datos relacionales lo entendía perfectamente. Tenían sus filas y sus campos y dentro guardábamos la información, pero... ¿Cómo puede almacenarse la información dentro de las bases de datos orientadas a objetos?", se pregunta Pilar, "¿qué ocurre con las relaciones entre tablas?". José, un poco extrañado no sabe qué responder a su compañera.

Nudo: ¿Cómo crees que se gestiona la información dentro de una base de datos orientada a objetos? ¿Existirá el concepto de clave primaria o de clave foránea?

Desenlace: Es normal que cuando nos topamos con las bases de datos orientadas a objetos no comprendamos muy bien de primeras cómo pueden funcionar. Venimos de estudiar una base de datos muy diferente, como es la relacional, que tiene unos conceptos muy claros como son el tema de la integridad de datos, las claves primarias y foráneas, relaciones entre tablas, etc.

En las bases de datos orientadas a objetos lo que se guarda son objetos, y entre éstos, no existen ningún tipo de relación equivalente a las de las bases de datos relacionales, es decir, aquí no existen claves primarias ni foráneas. Los objetos se irán almacenando dentro de la base de datos y su almacenamiento se parecerá más al de un fichero que al de una base de datos.


Fig. 4. Es importante comprender el funcionamiento de las bases de datos.

/ 5. Gestores de bases de datos orientadas a objetos

Cuando hablamos en unidades anteriores de las bases de datos relacionales, vimos que los programas que las gestionaban se llamaban Sistemas Gestores de Bases de Datos.

En esta unidad estamos tratando las bases de datos orientadas a objetos, y al igual que las anteriores, existen programas que son capaces de gestionarlas, los que conocemos como **Sistemas Gestores de Bases de Datos Orientadas a Objetos.**

Dentro de estos sistemas gestores de bases de datos orientadas a objetos podemos encontrarnos **dos tipos:**

a. **Puros:** Estos sistemas gestores de bases de datos únicamente soportan **bases de datos orientadas a objetos**.

b. **Mixtos:** Estos sistemas gestores de bases de datos soportan tanto **bases de datos orientadas a objetos como bases de datos relacionales.**

Estos sistemas gestores de bases de datos deben **cumplir dos condiciones:**

- **Deben ser un Sistema Gestor de Bases de Datos.** Es decir, deben poder tratar a los objetos que están almacenados como datos tal cual, no mostrando diferencias al usuario a la hora de tratar con ellos.
- **Debe soportar la orientación a objetos**. Estos sistemas deben poder soportar la orientación a objetos, ya que los datos que se están gestionando siguen este paradigma, y podrán emplear las características del mismo, como son la herencia, la ocultación de información, el polimorfismo, etc.

Existen muchos Sistemas Gestores de Bases de datos Orientadas a Objetos, como pueden ser:

- Oracle.
- Microsoft SQL Server.
- SQLite.
- PostgreSQL.
- Etcétera.

Nosotros vamos a usar el Sistema Gestor de Bases de Datos Orientadas a Objetos DB4O, debido a su simplicidad y a que es portable.

Audio 1. "SGBDOO"
http://bit.ly/2LIY940

/ 6. Conectores Java-DB4O

Los conectores de Java con DB4O son una **biblioteca que nos va a proporcionar las herramientas y métodos necesarios para poder conectar un programa escrito en Java con una base de datos orientada a objetos DB4O.**

Podemos descargar los conectores de la siguiente web:

https://sourceforge.net/projects/videomer/files/updates/0.1.0/db4o-8.0.184.15484.jar/download

Al entrar en la página de descarga deberemos esperar unos 5 segundos y se descargará la biblioteca de DB4O en **formato JAR.**

No es necesario **instalar ningún sistema gestor de bases de datos adicional**, ya que DB4O nos permite trabajar independientemente de la plataforma y directamente con la biblioteca descargada.

Cuando creemos la base de datos tendremos un **fichero binario** donde se guardarán todos los objetos de la misma, es decir, esta **base de datos** va a ser **portable.**

Hay que tener mucho cuidado ya que **si borramos por accidente** el fichero de la **base de datos la perderemos**, y no podremos recuperarla desde código.

El conector que acabamos de descargar nos provee de las siguientes **clases para la manipulación de información** en una base de datos orientada a objetos DB4O:

CLASE	FUNCIONALIDAD
ObjectContainer	Esta clase representa un contenedor para nuestros objetos de la base de datos, es decir, esta será nuestra base de datos orientada a objetos.
ObjectSet	Esta clase representa el contenedor de los objetos que se utilizará para las consultas, es decir, que al realizar consultas, tendremos un objeto de este tipo.

Tabla 1. Clases para acceso a base de datos DB4O.

/ 7. Agregar bibliotecas al proyecto

El siguiente paso una vez tengamos la biblioteca de DB4O descargada, es **agregarla a nuestro proyecto**, tal y como lo implementamos en la unidad anterior **con el conector para la base de datos relacional.**

Es decir, una vez creado nuestro nuevo proyecto, vamos a crear una **carpeta** llamada ***libraries*** (por ejemplo) en su interior, y nos llevamos ahí la biblioteca de DB4O.

Para poder agregar la biblioteca debemos pulsar con el botón derecho en la zona izquierda de NetBeans donde aparece *Libraries* o Bibliotecas y seleccionamos **Agregar archivo JAR.**

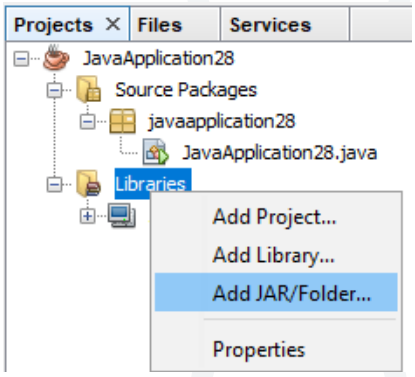

Fig. 5. Agregar biblioteca a nuestro proyecto.

Ahora **seleccionaremos dónde se encuentra la biblioteca que queremos agrega**r, es decir, navegaremos hasta la carpeta *libraries* que hemos creado en nuestro proyecto y la seleccionaremos.

Aquí es muy importante dejar **activa la casilla *Relative Path***, para que NetBeans agregue con la ruta relativa la biblioteca y no de error si movemos de sitio el proyecto.

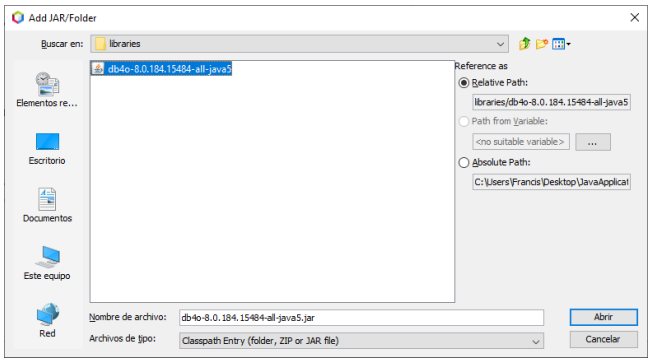

Fig. 6. Seleccionando la biblioteca.

Una vez hecho esto tendremos la biblioteca agregada a nuestro proyecto y podremos emplear sus funcionalidades.

/ 8. Conectar, desconectar e inserción de datos

Como ya vimos en la unidad anterior, cuando trabajamos con bases de datos, sea cual sea el modelo, vamos a tener **dos operaciones** que son **esenciales:** la **conexión y la desconexión** de la misma.

El primer paso a realizar siempre deberá ser **conectarse a la base de datos** deseada, para así poder realizar la acción requerida, y a continuación, desconectarnos.

Como ya sabemos, en resumen, siempre que vayamos a realizar una operación sobre una base de datos deberemos:

1. Conectar a la base de datos.
2. Realizar la operación pertinente.
3. Desconectar de la base de datos.

Para conectar con la base de datos orientada a objetos deberemos utilizar el siguiente código:

```
ObjectContainer bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "ruta fichero");
```

Código 1. Conectar con la base de datos.

En ella, observamos que tendremos que pasarle la **ruta del fichero** que será la base de datos orientada a objetos.

Para cerrar basta con llamar al **método *close()* del objeto *bd*.**

Estas operaciones lanzan varias excepciones que deberemos tratar con el bloque *try-catch* correspondiente.

Estas son:

```
Db4oIOException, DatabaseFileLockedException, IncompatibleFileFormatException,
OldFormatException y DatabaseReadOnlyException
```

Código 2. Bloque try-catch.

El cierre de la base de datos deberemos hacerlo en el **bloque *finally*,** para asegurarnos de que la base de datos se ha cerrado correctamente.

Para insertar datos deberemos usar el método *store* de la clase ***ObjectContainer***, al que le pasaremos un objeto.

Como es lógico, deberemos crear las clases necesarias para poder crear los objetos, pudiendo almacenar todos los que necesitemos.

```
Alumno alumno1 = new Alumno("Juan Pérez", 23,8.75);
bd.store(alumno1);
```

Código 3. Creación de objetos.

Fig. 7. Insertando datos.

/ 9. Consultas de datos

Seguro que ya conocemos de la asignatura de Bases de Datos que, **para recuperar datos de una base de datos relacional**, utilizamos **SQL.** Con las bases de datos **orientadas a objetos** utilizaremos otro lenguaje llamado **QBE** (*Query By Example* – Consulta según un ejemplo) el cual es muy diferente a SQL.

Para utilizar este lenguaje deberemos crear objetos "de ejemplo" para que QBE sepa lo que queremos. Es decir, que tendremos que crear un objeto del tipo de dato que queremos recuperar y poner todos sus atributos al valor por defecto que entiende QBE. Estos valores por defecto son:

- ***Datos de tipo int***: 0
- ***Datos de tipo double***: 0
- ***Datos de tipo String***: *null*

Vamos a ver **un ejemplo** de cómo podemos recuperar todos los datos de una clase *Alumno* que tiene como atributos el nombre y las tres notas de la evaluación de cada alumno, una por cada trimestre.

Lo primero deberá ser que, dentro de la clase Alumno**, crear de forma obligatoria el constructor con todos los parámetros**, los **métodos *get* y *set,* y el método *toString***, ya que este último lo utilizará QBE más adelante.

Visto esto, para recuperar todos los objetos de tipo *Alumno* de la base de datos orientada a objetos mediante una consulta QBE, tendremos **el código** que se muestra en la siguiente figura.

```
ObjectContainer bd = null;

try {
    bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "alumnos.db4o");
    Alumno ejemplo = new Alumno(null, 0, 0, 0);
    ObjectSet res = bd.queryByExample(ejemplo);
    while (res.hasNext()) {
        System.out.println(res.next());
    }
} catch (Exception ex) {
    System.out.println("Error: " + ex.toString());
} finally {
    bd.close();
}
```

Fig. 8. Recuperando datos I.

En el código podemos ver cómo **creamos el alumno de ejemplo**, y mediante el **método *queryByExample*** de la clase *ObjectContainer,* ejecutamos la consulta QBE que devuelve un objeto del tipo *ObjectSet* con todos los alumnos que contenga la base de datos.

Para mostrar **el resultado** recorremos el ***ObjectSet*** mediante un **bucle *while*** haciendo uso del **método *hastNext(),*** que nos indica si hemos llegado al final del resultado objeto o no, y mostramos por pantalla el objeto obtenido mediante el método *next()*, el cual, hará uso el método *toString* de la clase *Alumno* para mostrar la información por pantalla.

9.1. Consultas de datos con condiciones

Ya hemos aprendido a obtener todos los objetos de un tipo concreto de la base de datos, pero ¿cómo podemos obtener objetos que cumplan ciertas condiciones? Como sabemos, el lenguaje QBE se basa en ejemplos, por tanto, si queremos obtener todos los alumnos con una nota de 7 en el primer trimestre, tendremos que **crear un ejemplo para ello** de la siguiente forma:

```
Alumno ejemplo = new Alumno(null, 7, 0, 0);
```

Código 4. Obtención de objetos que cumplan una condición.

Como podemos observar, los campos de nombre, nota2 y nota3 los deberemos dejar a su valor por defecto para que no interfieran en la restricción, pero el campo nota1, lo hemos puesto a un valor de 7. Con este ejemplo QBE entiende que queremos obtener todos los alumnos cuya nota1 sea igual al valor 7.

En el caso de que queramos **unir más de una condición**, como cuando en SQL usábamos las instrucciones *AND* y *OR* en las consultas, y permitían concatenar más de una condición, con QBE, para poder concatenar más de una condición a la vez, es tan sencillo como **dar el valor que queramos a los atributos de la clase**.

Por ejemplo, si queremos obtener de la base de datos todos los alumnos que tengan un 7 en el primer trimestre, un 8 en el segundo y un 9 en el tercero deberemos crear el siguiente alumno de ejemplo:

```
Alumno ejemplo = new Alumno(null, 7, 8, 9);
```

Código 5. Obtención de objetos que cumplan una condición (II).

El lenguaje de consultas QBE **es muy limitado** y solo nos va a permitir **realizar consultas donde busquemos valores iguales a otro**, pero no podremos buscar utilizando operadores de mayor, mayor o igual, menor, menor o igual o distinto, es decir, no podremos buscar, por ejemplo, todos los alumnos que hayan aprobado los tres trimestres, ya que no podremos hacer una búsqueda donde las notas sean mayor o igual a 5.

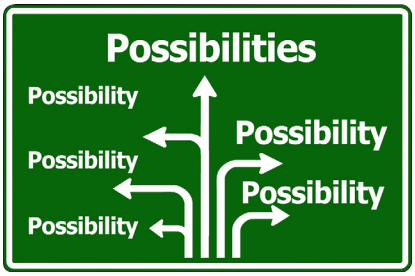

Fig. 9. Consultas con condiciones.

Vídeo 1. "Consultando datos con QBE"
https://bit.ly/39llBg3

/ 10. Actualización de datos

En una base de datos relacional, para actualizar datos utilizamos la sentencia *UPDATE* de *SQL*.

Actualizar objetos de la base de datos orientada a objetos utilizando el lenguaje QBE es muy simple, basta con **obtener el objeto que se desea actualizar, modificarlo y volverlo a insertar en la base de datos con la función *store*() para que se actualice correctamente.**

Esta nueva inserción hará uso del control de versiones que soporta DB4O y actualizará automáticamente el objeto, de forma que cuando volvamos a recuperarlo tendrá los datos actualizados.

Por ejemplo, imaginemos que el alumno Pedro tenía el primer trimestre suspenso con un 4 y lo ha conseguido recuperar, obteniendo una nota de 5. Para poder actualizarle la nota correctamente deberemos escribir:

```
Alumno ejemplo = new Alumno("Pedro", 5, 0, 0);
ObjectSet res = bd.queryByExample(ejemplo);
while (res.hasNext())
{
        Alumno nuevo = (Alumno) res.next();
        nuevo.setNota(nota);
        // Volvemos a guardar el alumno en la BDOO para que se actualice
        bd.store(nuevo);
}
```

Código 6. Actualizar objetos de la base de datos.

Con esta forma de actualización de información hay que tener mucho cuidado, ya que si existieran dos alumnos que se llamasen igual les modificaría la nota a los dos.

Fig. 10. Actualizando datos.

Así que para el uso de esta funcionalidad tendremos que utilizar de forma obligatoria **campos de los objetos que estemos seguros que no se van a poder repetir,** para así intentar emular a las claves primarias.

/ 11. Caso práctico 2: "¿Se puede seguir usando SQL?"

Planteamiento: Nuestros amigos están haciendo sus primeras consultas en una base de datos orientada a objetos. Estas consultas les resultan un poco extrañas. José le pregunta a Pilar si cree que podrán usar SQL para operar con las bases de datos orientadas a objetos, ya que es mucho más fácil y lo entiende mejor. Pilar se queda mirándolo y le contesta que ella también lo preferiría, pero no sabe darle una respuesta.

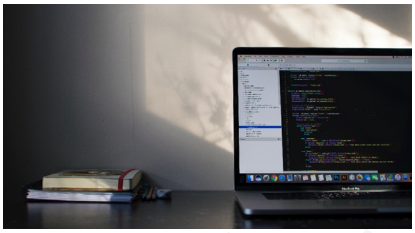

Fig. 11. Bases de datos y programación van de la mano.

Nudo: ¿Qué piensas al respecto? ¿Crees que se podrán utilizar el lenguaje SQL para poder operar con bases de datos orientadas a objetos?

Desenlace: Como sabemos, existen muchos gestores de bases de datos que soportan las bases de datos orientadas a objetos.

Algunos de éstos tienen su propio lenguaje para el acceso a los datos, como es el caso que nos ocupa en esta unidad, el DB4O, pero existen gestores de bases de bases de datos como Oracle y Microsoft SQL Server que además de trabajar con bases de datos relacionales soportan las bases de datos orientadas a objetos. En estos casos sé existe una especie de extensión de SQL, que permite trabajar con objetos, aunque no es exactamente igual que el SQL que trabaja con bases de datos relacionales.

/ 12. Borrado de datos

En una base de datos relacional, para borrar datos utilizamos la sentencia *DELETE* de *SQL*.

Borrar objetos de la base de datos orientada a objetos utilizando el lenguaje QBE, al igual que ocurría con la actualización, se puede hacer de forma muy simple, ya que **basta con obtener el objeto que se necesita borrar, y eliminarlo de la base de datos con la función *delete()*.** Esto hará que se borre de forma permanente el objeto que le pasamos a la función *delete*.

Por ejemplo, imaginemos que queremos eliminar todos los alumnos cuya nota en el tercer trimestre sea de un 5. Para ello deberemos escribir:

```
Alumno ejemplo = new Alumno(null, 0, 0, 5);
ObjectSet res = bd.queryByExample(ejemplo);
while (res.hasNext())
{
        Alumno nuevo = (Alumno) res.next();
        bd.delete(nuevo);
}
```

Código 7. Borrar objetos de la base de datos.

Este código obtendrá todos los alumnos cuya nota en el trimestre 3 sea un 5 y los irá eliminando uno a uno.

Igualmente, como en otras operaciones de QBE, con esta forma de borrado de información hay que tener mucho cuidado, ya que podremos eliminar objetos que no deseamos borrar, así que, para el uso de esta funcionalidad, tendremos que **utilizar de forma obligatoria campos de los objetos que estemos seguros que no se van a poder repetir**, para así intentar emular a las claves primarias.

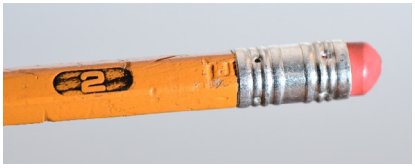

Fig. 12. Borrado de datos.

Vídeo 2. "Borrando datos con QBE"
https://bit.ly/2XrdIA6

/ 13. Aplicando la POO

Al igual que con las bases de datos relacionales de la unidad anterior, **vamos a crear una clase que nos gestione todo el proceso de la conexión con la base de datos orientada a objetos con DB4O**.

1. En primer lugar, vamos a **crear una excepción** llamada ***DB4OException*,** la que utilizaremos para el tratamiento de excepciones, evitando así tener muchas excepciones y centrarnos solo en una.

2. Siguiendo la filosofía de la programación orientada a objetos, vamos a **crear una clase para gestionar la conexión con la base de datos**, aprovechando así al máximo la POO.

   Así, podemos crear la clase ***ConexionDB4O*** que constará de los siguientes métodos:

   - **Un constructor** al que le pasemos la ruta del fichero que actuará como base de datos.
   - **Un método *conectar*,** que conectará con la base de datos. Este método lanzará una excepción *DB4OException*.
   - **Un método *desconectar*,** que desconectará de la base de datos. Este método lanzará una excepción *DB4OException*.

Si recordamos, en la unidad anterior teníamos un método para ejecutar las consultas *SELECT* y otro para el resto.

   En este caso, como no es tan sencillo utilizar el lenguaje QBE tendremos que **crear todos los métodos de consulta, borrado y actualización de datos en esta clase.**

   Todos los **métodos de consulta** de datos deberán **devolver un *array* del objeto que recuperan**, mientras que los **métodos de insertado, borrado y actualización** podrán **devolver un entero con los objetos afectados,** de forma análoga a como lo hacíamos en la gestión de la conexión a MySQL.

3. En el apartado de recursos del tema, podrás encontrar ejemplos de clases para llevar a cabo la conexión, así como la ***DB4OException*** comentada.

Fig. 13. Si aprovechamos la POO todo será más fácil.

/ 14. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad hemos visto qué son las **bases de datos orientadas a objetos** y sus características.

Hemos comprobado que existen **gestores de bases de datos** que las soportan y que no necesitamos tener instalado uno de ellos para poder usarlas desde Java.

También hemos aprendido a instalar correctamente la **biblioteca** que nos permitirá hacer uso de este tipo de base de datos.

Por último, hemos estudiado las operaciones que podemos hacer sobre este tipo de bases de datos, cómo **insertar** objetos, cómo podemos **consultar** objetos de la base de datos con ciertas restricciones, cómo podemos **actualizar** los datos de la misma y cómo podemos **borrar** los datos que ya no necesitemos.

Resolución del caso práctico de la unidad

Con respecto a las bases de datos, ya sabemos que existen multitud de tipos diferentes a las relacionales. Una de ellas es la que hemos estudiado en el tema: las orientadas a objetos.

Cuando conocemos varios tipos de bases de datos nos surge la pregunta de cuándo tenemos que utilizar un tipo u otro.

El uso de un tipo de base de datos o de otra, va a depender casi exclusivamente de la aplicación a desarrollar, por ejemplo, en el caso de un programa que gestione la información que se produce de un centro educativo, lo más normal sería usar una base de datos relacional, ya que nos facilitará en gran medida las operaciones que tengamos que realizar (no obstante, habría que tener cuidado, ya que si tenemos un volumen muy elevado de datos a gestionar, las bases de datos relacionales pueden volverse lentas)

En el caso de las bases de datos orientadas a objetos, se usan principalmente cuando no hay una gran cantidad de información a gestionar, y necesitamos una base de datos portable independiente de la plataforma.

Fig. 14. Las elecciones son importantes.

/ 15. Bibliografía

Colaboradores de Wikipedia. (2020, mayo 25). *Base de datos - Wikipedia, la enciclopedia libre*. Recuperado 3 de junio de 2020, de https://es.wikipedia.org/wiki/Base_de_datos

Colaboradores de Wikipedia. (2019, diciembre 21). *Base de datos orientada a objetos - Wikipedia, la enciclopedia libre*. Recuperado 3 de junio de 2020, de https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_objetos

Isabelvalera55, M. (s. f.). *Programación orientada a objetos. Oracle y SQL Server. - Monografias.com*. Recuperado 3 de junio de 2020, de https://www.monografias.com/trabajos4/basesdatos/basesdatos.shtml