Centro Oficial FP Digital & Tech

ACCESO A BASES DE DATOS ORIENTADA A OBJETOS.

1. MANTENIMIENTO DE LA PERSISTENCIA DE LOS OBJETOS. BASES DE DATOS ORIENTADAS A OBJETOS.

La persistencia de los datos se refiere a la capacidad de almacenar información de forma permanente y duradera. En el ámbito de la informática y la programación, se utiliza para describir la habilidad de un sistema o una aplicación de retener los datos incluso después de que el programa que los generó haya dejado de ejecutarse o el sistema se haya reiniciado.

Hasta ahora hemos visto cómo hacer esto guardando la información en ficheros pero ¿qué ocurre si queremos almacenar grandes cantidades de información? Usar un fichero se vuelve inviable. Para ello usaremos las bases de datos orientadas a objetos BDOO.

Una BDOO, es una base de datos diseñada específicamente para almacenar y manipular datos en forma de objetos, es decir, que en estas bases de datos se pueden almacenar objetos completos con sus atributos y sus métodos. A diferencia de lo que ocurre con la distribución en tablas de otras bases de datos, en las BDOO toda la información está disponible en el objeto y no repartida en filas y columnas.

Los objetos a su vez se dividen en clases, creando así una jerarquía de clases y subclases, en la que las subclases heredan las propiedades de las clases superiores y las complementan con sus propios atributos. Pero esta jerarquía no es estricta, puesto que los objetos de una clase pueden relacionarse con otras clases, formando redes. Así mismo, los objetos simples pueden combinarse para crear objetos más complejos.

El sistema de gestión de base de datos orientadas a objetos asigna de forma automática un código de identificación único a cada registro (que será inmutable), permitiendo así recuperar los objetos cuando estos se han guardado y realizar consultas.

En estas bases de datos, los objetos tienen relaciones de varios a varios, en las que los indicadores (que son invisibles al usuario, puesto que funcionan a nivel interno) son los encargados de establecer dichas relaciones entre objetos.

Por último, recalcar que las BDOO no emplean lenguaje SQL, sino lenguajes de programación.

2. CARACTERÍSTICAS DE LAS BDOO.

Algunas características son:

- **Persistencia de objetos**: Las BDOO permiten almacenar objetos directamente en la base de datos, conservando su estado y estructura. Esto significa que los objetos pueden mantenerse en la base de datos incluso después de que la aplicación que los creó haya finalizado.
- **Consultas y manipulación de objetos:** Las BDOO proporcionan un lenguaje de consulta y manipulación de objetos, que permite realizar operaciones como recuperar objetos, modificar sus atributos, buscar objetos según ciertos criterios y establecer relaciones entre ellos. Esto facilita el acceso a los datos de manera más intuitiva y eficiente.
- **Herencia y polimorfismo**: Las BDOO admiten la herencia entre objetos, lo que significa que un objeto puede heredar atributos y comportamientos de otro objeto. Además, permiten el polimorfismo, lo que permite tratar objetos de diferentes clases de manera uniforme. Estas características fomentan la reutilización de código y la representación más precisa de las relaciones del mundo real.
- **Transacciones y concurrencia**: Las BDOO ofrecen soporte para transacciones, lo que garantiza la integridad y consistencia de los datos al realizar operaciones múltiples como una unidad atómica. También manejan la concurrencia, permitiendo que múltiples usuarios accedan y modifiquen los datos simultáneamente sin generar conflictos.

Si comparamos estás con la relacionales:

	BDOO	RELACIONALES
Modelo de datos	Objetos con atributos y métodos => **Modelado de "mundo real"**	Filas y columnas
Estándar	**Carece de estándares.**	Estandarizada.
Flexibilidad	Permiten representar objetos complejos y relaciones. Fácil agregar cosas sin cambiar esquema	Estructura fija, cambiar esquema para agregar atributos

Persistencia	Objetos más fácil	Objetos y relaciones complicado
Consultas y manipulación	Operaciones sobre los objetos de manera intuitiva y eficiente => **Alta velocidad de procesamiento**	Las consultas se basan en álgebra y cálculo relacional.
Herencia y polimorfismo	Admiten la herencia y el polimorfismo	No tienen soporte nativo para la herencia y el polimorfismo

Tabla 1

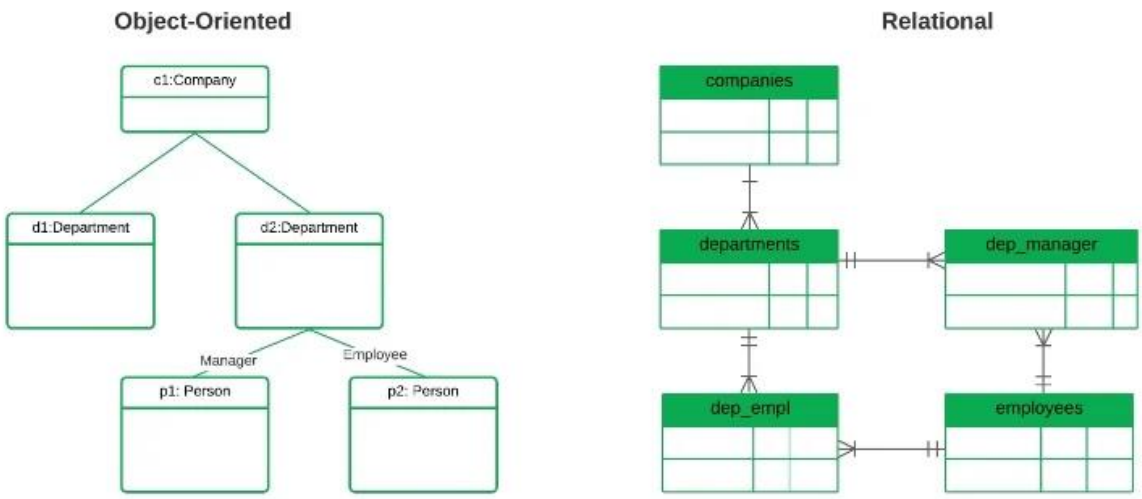

Imagen 1.

2.1 Instalación del gestor de bases de datos.

Para trabajar con BDOO en Netbeans vamos a usar el gestor de base de datos DB4O. Para instalarlo vamos a seguir los siguientes pasos.

- Descargar la librería db4o => db4o-java5-5.5.1.jar
- Instalar la librería en nuestro proyecto => Vamos a nuestro proyecto, "*Libraries"* botón derecho, "*Add JAR/Foder*".

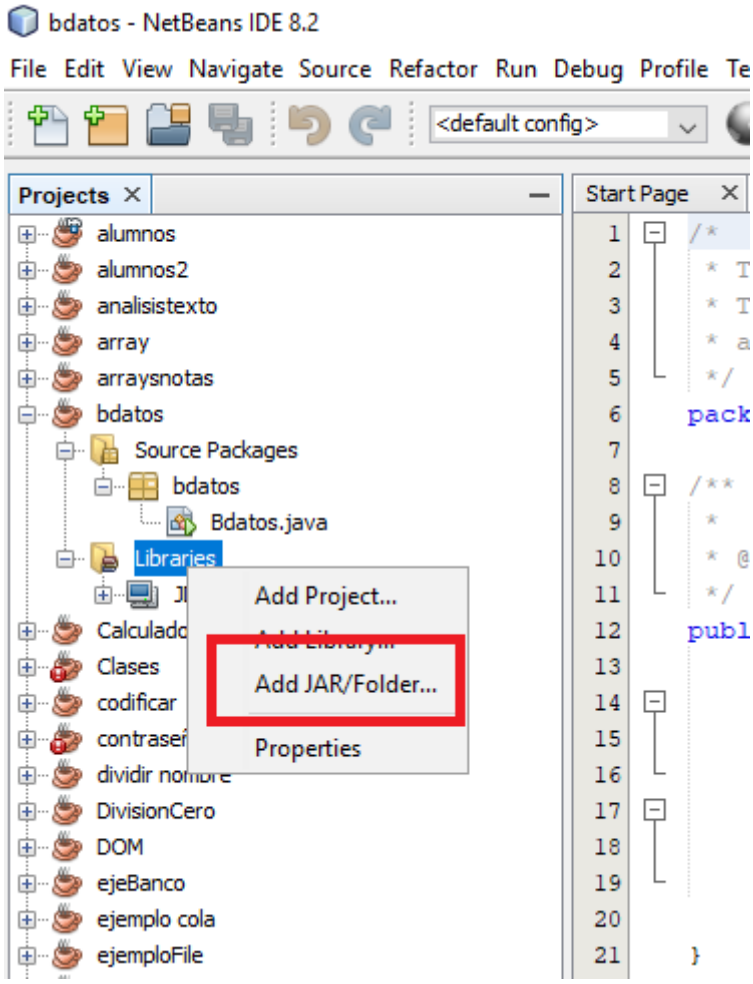

Imagen 2

- Buscar y seleccionar el .jar de la versión db4o:

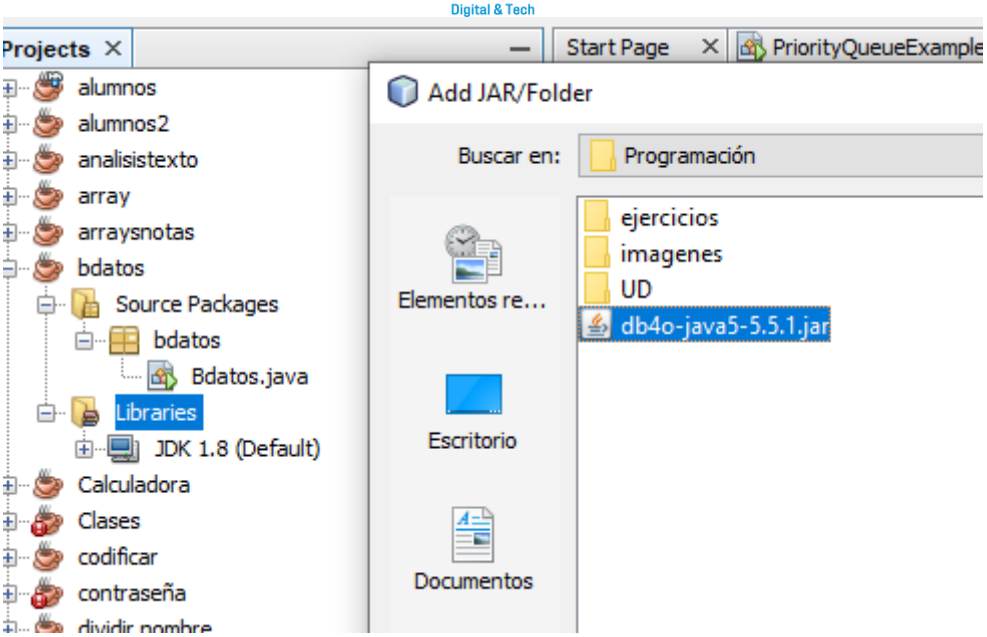

Imagen 3

La librería del proyecto quedaría de la siguiente manera:

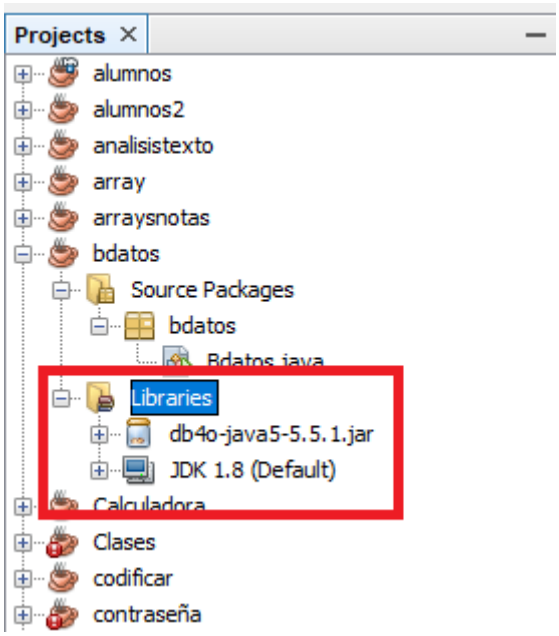

Imagen 4

2.2 Creación de bases de datos.

Una vez integrada la librería db4o en nuestro sistema usaremos las clases y métodos que nos ofrece para crear nuestra base de datos.

Db4o es una BDOO embebida, es decir, que se integra directamente en la aplicación y utiliza archivos locales para almacenar los objetos persistentes. Dicho coloquialmente, la tendremos como un fichero binario de nuestro proyecto y, por tanto, para crearla necesitaremos “crear/abrir” este fichero.

```
import com.db4o.Db4o;
import com.db4o.ObjectContainer;


public class Example {
   public static void main(String[] args) {
      // Abrir/crear la base de datos.
      ObjectContainer db = Db4o.openFile("database.db4o");

      // Resto del código...

      // Cerrar la base de datos
      db.close();
   }
}
```

NOTA: A partir de la versión 8.0 de db4o, se introdujo una nueva forma de trabajar con la base de datos utilizando la clase Db4oEmbedded que permite configuraciones personalizadas de la BDOO.

Analizando el ejemplo anterior vemos que hay que crear un objeto del tipo *ObjectContainer.* Esta clase es de la librería db4o por lo que tendremos que importarla primero.

Un contenedor de datos es una estructura que permite almacenar y organizar información de manera ordenada. Imagina que es una especie de caja o recipiente donde puedes guardar diferentes tipos de objetos o valores. Ya hemos visto contenedores de datos, aunque no se hayan llamado de tal manera: listas, colas…

En db4o vamos a usar contenedores para manejar nuestra información, digamos que nos va a permitir hacer las consultas a la BD con los distintos métodos que tiene este contenedor. Por tanto, para trabajar con la base de datos db4o se necesita “introducir” sus elementos en el contenedor, luego se usarán los métodos de esta clase para trabajar con los elementos, finalmente, será necesario usar el método *close* para cerrar la BD. Veamos algunos de los métodos más usados:

- **void store(Object objeto)**: El objeto pasado como argumento se guarda en la base de datos y se le asigna un identificador único. NOTA: En la que

db4o 5.5 que tenemos nosotros es **set y luego commit()** para que los cambios se guarden en la base de datos.

- **void delete(Object objeto)**: Este método se utiliza para eliminar un objeto de la base de datos.

- **ObjectSet<Object> queryByExample(Object objeto)**: La base de datos busca todos los objetos que coincidan con el ejemplo y los devuelve como un conjunto de resultados (ObjectSet) que puedes iterar para acceder a cada objeto. NOTA: En la que db4o 5.5 que tenemos nosotros es **query(Objeto objeto).**

- **Object getByID(int id)**: Este método se utiliza para recuperar un objeto de la base de datos según su identificador único. Se proporciona el identificador como argumento y se devuelve el objeto correspondiente si existe en la base de datos.
- **void close()**: Este método cierra la conexión con la base de datos. Es importante llamar a este método al finalizar las operaciones para liberar los recursos y garantizar la integridad de los datos.

Supongamos la clase Persona:

```
public class Persona {

    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public int getEdad() {
        return edad;
    }
    public String getNombre() {
        return nombre;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    @Override
```

```
    public String toString() {
        return "Persona{" + "nombre=" + nombre + ", edad=" + edad + '}';
    }
}
```

Creamos un main donde tenemos una base de datos, introducimos los objetos y luego los recuperamos:

```
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;


public class Bdatos {


    public static void main(String[] args) {
        //Creo/abro la base de datos
        ObjectContainer db = Db4o.openFile("database.db4o");

        //Creo los objetos personas
        Persona persona1 = new Persona("Manuel", 25);
        Persona persona2 = new Persona("Elena", 30);

        //Los almacenamos en la BD
        db.set(persona1);
        db.set(persona2);
        db.commit();



        // Recuperar todos los objetos de la clase Persona
        ObjectSet resultados = db.query(Persona.class);



        //mientras haya algo en la colección lo imprimimos
        while (resultados.hasNext()) {
            System.out.println(resultados.next());
        }


        // Cerrar la conexión con la base de datos
        db.close();
    }
```

}

Si observamos el ejemplo anterior vemos que para recuperar los objetos necesitamos una colección propia de db4o: **ObjectSet,** esta colección guardará objetos genéricos, y tiene los siguientes métodos para su uso:

- **boolean hasNext()**: Devuelve true si hay más objetos en el conjunto, es decir, si se puede llamar al método next() para obtener el siguiente objeto.

- **Object next()**: Devuelve el siguiente objeto en el conjunto y mueve el puntero al siguiente objeto.
- **long size()**: Devuelve la cantidad de objetos en el conjunto.
- **boolean isEmpty()**: Devuelve true si el conjunto está vacío, es decir, no contiene objetos.
- **void close()**: Cierra el conjunto y libera los recursos asociados.
- **void reset()**: Restablece el puntero del conjunto a la posición inicial, lo que permite recorrer los objetos nuevamente desde el principio.
- **void remove(Object obj)**: Elimina un objeto específico del conjunto.

Hay que tener en cuenta que db4o tiene sus propias excepciones y que habrá que recogerlas cuando sea necesario:

- com.db4o.ext.**Db4oIOException**: Esta excepción se lanza cuando ocurre un error de E/S durante las operaciones de lectura o escritura en la base de datos.
- com.db4o.ext.**DatabaseClosedException**: Esta excepción se lanza cuando intentas realizar una operación en una base de datos que está cerrada.

- com.db4o.ext.**DatabaseReadOnlyException**: Esta excepción se lanza cuando intentas realizar una operación de escritura en una base de datos abierta en modo de solo lectura.

- com.db4o.ext.**ObjectNotStorableException**: Esta excepción se lanza cuando intentas almacenar un objeto que no es válido para ser almacenado en la base de datos.

- com.db4o.ext.**IncompatibleFileFormatException**: Esta excepción se lanza cuando intentas abrir un archivo de base de datos con una versión incompatible de Db4o.

2.3 Mecanismos de consulta.

Para realizar las consultas a la BD usaremos el método query() del contenedor de objetos. En el ejemplo anterior hemos usado de parámetro la clase que queríamos que nos buscara, pero si queremos hacer una consulta más compleja usaremos el método de la siguiente manera:

Query query()

Es decir, este método nos devolverá un objeto Query que representa la consulta a realizar. Este objeto tiene una serie de métodos que nos permitirá hacer una consulta más compleja. No hay que olvidar hacer el **.execute()** al terminar para que se ejecute la consulta y se guarde en un ObjectSet.

- **constrain( )** restringe la búsqueda.
- **descend(String atributoName):** busca entre el atributo dado.
- **orderAscending():** ordena de manera ascendente.
- **orderDescending():** ordena de manera descendente
- **execute():** ejecuta la consulta y la mete en un ObjectSet

Veamos un ejemplo, aplicado al código anterior:

```
Query query = db.query();
query.constrain(Persona.class);
query.descend("edad").constrain(30);
ObjectSet resultados = query.execute();
```

El método constrain() se usa para restringir la consulta a esos valores, y descend() nos permite seleccionar un atributo. Otro ejemplo, para ordenarlos por orden ascendente de edad.

```
Query query = db.query();
query.constrain(Persona.class);
query.descend("edad").orderAscending();
ObjectSet resultados = query.execute();
```

¿Y qué ocurre si quiero hacer consultas más específicas? Pues al usar el método constrain() podemos concatenarlo con otros métodos,como:

- greater(): mayor que.
- smaller():menor que.
- or(Constrain c): o
- and(Constrain c): y
- equals():

OJO: nótese que los métodos or() y and() necesitan que se les pase un objeto constrain, es decir, una consulta.

Veamos un ejemplo de consulta con estos métodos:

```
Query query = db.query();
query.constrain(Persona.class);

Constraint c1 =query.descend("edad").constrain(60).equal();
query.descend("edad").constrain(50).or(c1);
ObjectSet resultados = query.execute();
```

En este ejemplo hacemos una consulta de todas los objetos persona cuya edad sea 50 o 60.

2.4 Modificación de los datos en la BD.

Hasta ahora hemos introducido elementos en la la BDOO, y hecho consultas sobre ellos. Pero ¿cómo hacemos para modificar los datos o borrarlos? Pues ambas operaciones son igual de sencillas de hacer:

- Borrar. Bastará con usar el método delete(Object o) sobre nuestro contenedor.

- Modificar un elemento. Lo buscaremos en la BD, cambiaremos el valor que queramos modificar con un set y lo volveremos a introducir en la BD. Si lo vemos en un ejemplo:

```
ObjectContainer db = Db4o.openFile("database.db4o");

//Buscamos todos los conductores que se llamen Juan
 Query query = db.query();
query.constrain(Conductor.class);
query.descend("nombre").constrain("Juan").equal();
ObjectSet resultados=query.execute();


//modificamos su nombre a Pepe
while (resultados.hasNext()) {
    Conductor cond1 = (Conductor) resultados.next();
    cond1.setNombre("Pepe");
    db.set(cond1);
    db.commit();
}
```

2.5 Exportación de la base de datos.

Como ya hemos visto las BDOO no están estandarizadas, de manera que si exportamos/importamos una base de datos hecha con una aplicación para usar en otra no bastará con crear clases exactamente iguales ya que las consultas no reconocerían esas nuevas clases por muy iguales que sean a las otras. Por tanto, habrá que exportar la base de datos .db4o creada y, además, deberemos hacer un paquete con las clases usadas e importar este.

Veamos cómo exportar una librería creada por nosotros:

1. Crea la librería:

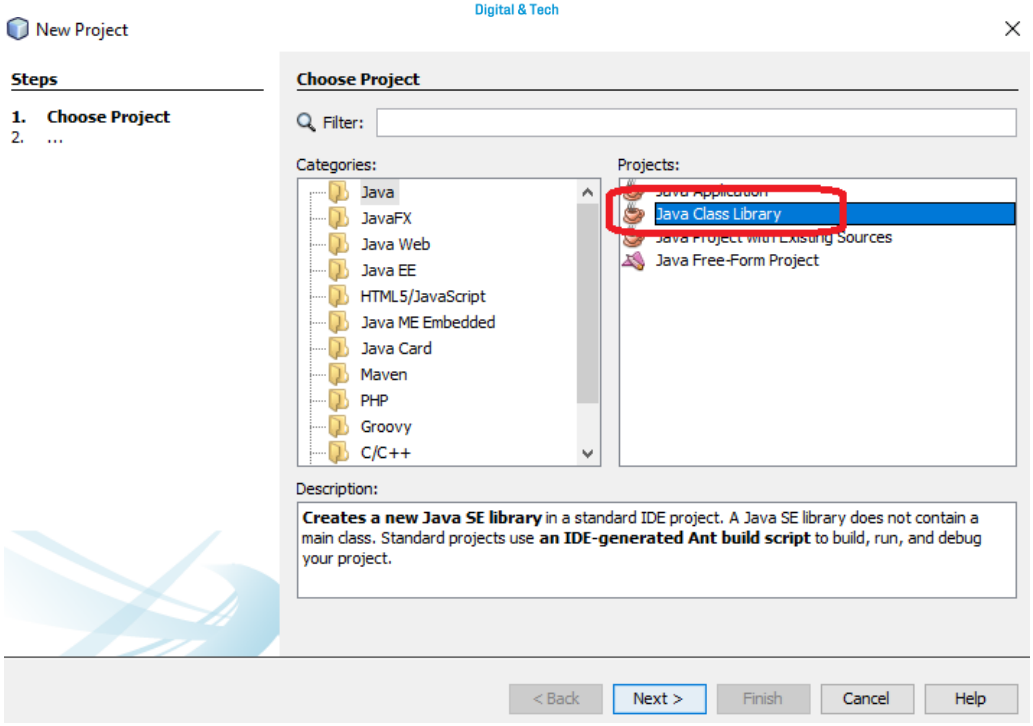

Imagen 5

2. Crea los paquetes y clases que necesites:

Imagen 6

3. Construye el proyecto:

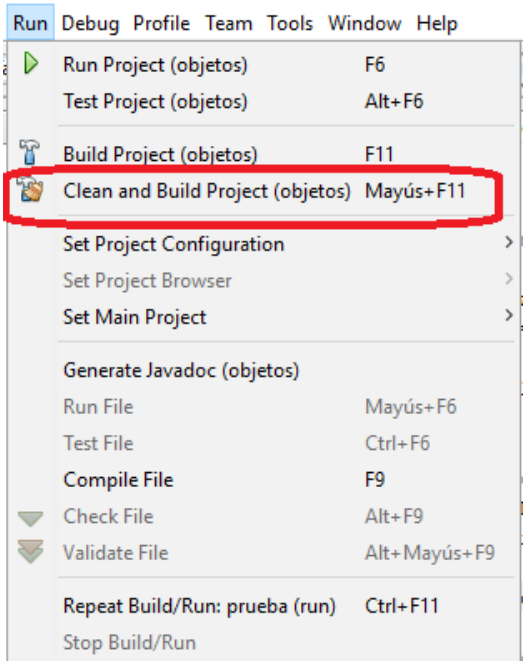

Imagen 7

4. Ya tendrás .jar en la carpeta dist del proyecto. Solo tendrás que importar esta librería junto con tu base de datos y la librería db4o.