
Ra-Ma®
Juan Carlos Moreno

# 9.1 BASES DE DATOS ORIENTADAS A OBJETOS

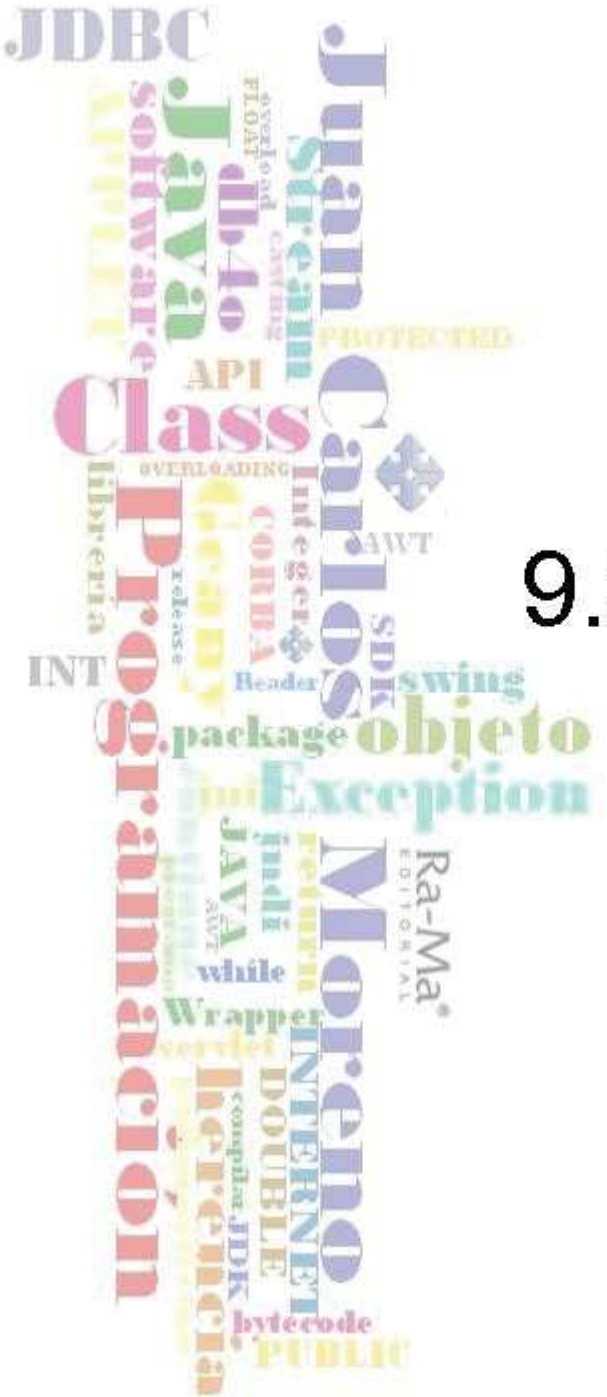
# BASES DE DATOS ORIENTADAS A OBJETOS



- Están especialmente diseñadas para trabajar con datos de tipo *objeto*
- Los modelos Orientados a Objetos los datos manejados por la base de datos serán clases y objetos.
- Al programar si se utilizan modelos relacionales se pierde parte de las ventajas de trabajar con Orientación a Objetos puesto que hay que hacer una transformación entre objetos y datos relacionales (y viceversa).
- ¿Por qué perder tiempo en rehacer las estructuras de datos (objetos) cuando se pueden almacenar y recuperar directamente?

# 9.1.1 BASES DE DATOS ORIENTADAS A OBJETOS COMERCIALES

- **Objetivity/DB**. Ofrece soporte para Java, C++, Python y otros lenguajes. Ofrece un alto rendimiento y escalabilidad. No es un producto libre.
- **db4o**. Es una Base de Datos Open Source para Java y .NET. Se distribuye bajo licencia GPL.
- **Intersystems Cache®.** Es una Base de Datos Orientada a Objetos que ofrece soporte para Java, C++, .NET, etc.
- **EyeDB**. Sistema basado en la especificación ODMG el cual está desarrollado y soportado por la compañía francesa SYSRA. Se distribuye bajo la licencia GNU (*GNU lesser General Public License*). Es un software libre.




Ra-Ma®
Juan Carlos Moreno

# 9.2 CARACTERÍSTICAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

# CARACTERÍSTICAS DE LAS BDOO (I)



- Sistema de modelado del mundo real.
- Cada entidad del mundo real será un objeto en la base de datos.
- Los objetos tienen un identificador único que los identifica y los diferencia de los demás objetos del mismo tipo.
- Pueden existir objetos con la misma información pero con diferente identidad.
- Es posible almacenar objetos complejos en la base de datos sin que por ello haya que realizar operaciones especiales sobre la base de datos.

# CARACTERÍSTICAS DE LAS BDOO (II)



- El concepto de herencia se mantiene incluso en la base de datos.
- Es el usuario el que modela los objetos de la base de datos y etiqueta los atributos y métodos que son visibles en la interfaz del objeto y cuáles no.
- El SGBDOO se encarga de acceder a los miembros de los objetos sin necesidad de escribir métodos para acceder a ellos.
- Acceso rápido a los datos dado que no hace falta realizar *joins* de tablas.
- Control de versiones. Algunos SGBDOO permiten un control de versiones.
- Implantación de conceptos del modelo OO como (polimorfismo, sobrecarga sobrescritura, etc.).

# **Ventajas** de las BDOO



- No tener que reensamblar los objetos cada vez que se accede a la base de datos.
- Cuando cambia un objeto la forma de actualizarlo en la base de datos es simplemente almacenándolo.
- La reutilización que es una de las características de los lenguajes de programación orientados a objetos se mantiene con lo que se mejoran los costes de desarrollo.
- El acceso a la información de forma natural.
- El control de acceso y concurrencia se facilita enormemente.
- Estos sistemas funcionan de forma eficiente en entornos cliente/servidor y arquitecturas distribuidas.
- No hace falta redefinir las relaciones entre objetos.

# **Limitaciones** de las BDOO

- El SQL es un estándar bastante asentado y fuerte.
- Estructura más compleja que las bases de datos relacionales.
- A veces es más eficiente almacenar los datos en bases de datos relacionales debido a las consultas que se van a realizar sobre ellas.
- Se reduce la velocidad de acceso.
- Cuando se trata de realizar consultas complejas resulta más adecuado una base de datos relacional accedida mediante SQL.

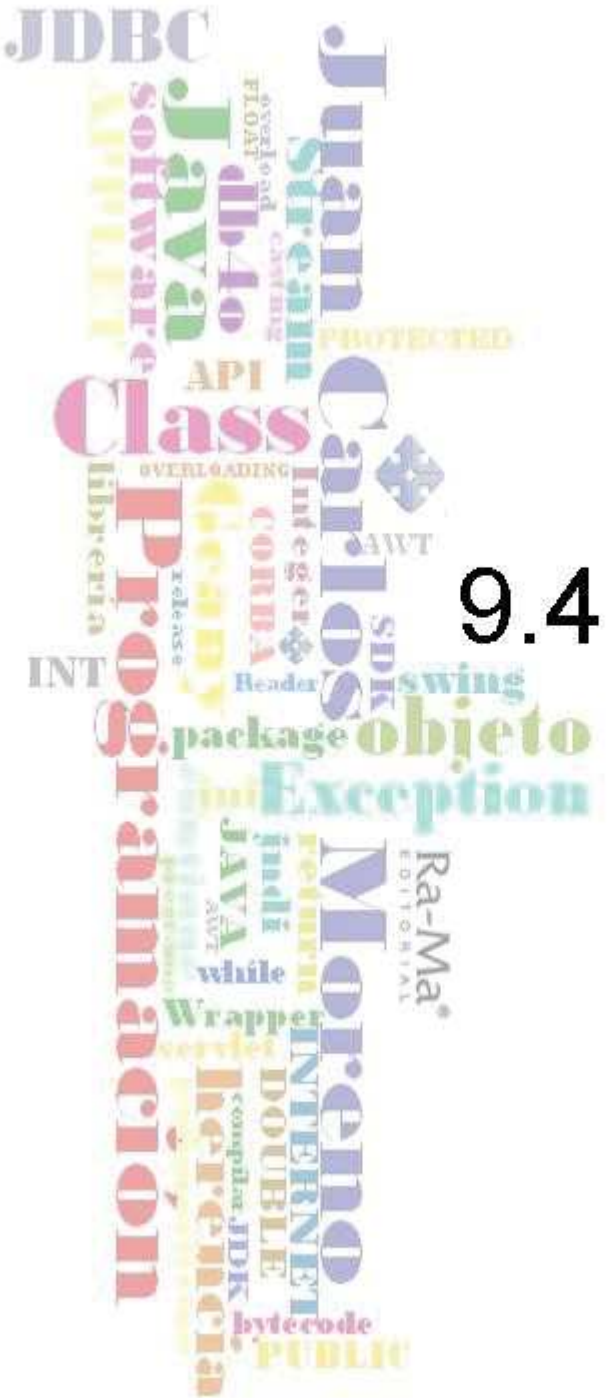

Ra-Ma®
Juan Carlos Moreno

# 9.3 INSTALACIÓN DEL GESTOR DE BASES DE DATOS

# EL GESTOR DE BASES DE DATOS db4o



- base de datos con licencia **GPL**.
- Db4o es una verdadera base de datos de objetos.
- El contenido de los objetos, así como la estructura y las relaciones son almacenados en la base de datos sin importar la complejidad que tenga la clase.
- Existen distintas distribuciones de la base de datos db4o para Java, .NET y Mono.




Ra-Ma®
Juan Carlos Moreno

# 9.4 EL API (APPLICATION PROGRAM INTERFACE)

# com.db4o.ObjectContainer

- Representa una base de datos en modo monousuario (*stand-alone*) o un cliente de un servidor db4o.
- Esta interface proporciona métodos para almacenar, consultar y borrar objetos así como realizar transacciones sobre la base de datos.
- Cada ObjectContainer representa una transacción. Todas las operaciones realizadas en la base de datos se hacen en modo transaccional de tal manera que cuando se haga commit() o rollback(), automáticamente se inicia la siguiente transacción.
- Cada ObjectContainer mantiene sus propias referencias a los objetos instanciados y almacenados.


Ra-Ma®
Juan Carlos Moreno

# 9.5 OPERACIONES BASICAS CON LA BASE DE DATOS

# CREAR/ACCEDER A LA BASE DE DATOS

```
ObjectContainer bd =
    Db4oEmbedded.openFile(Db4oEmbedded.new
    Configuration(),"alumnos.db4o");
try {
//Realizar operaciones o
//llamadas a métodos
}
finally {
bd.close();
}
```

# Métodos de apertura y cierre de una base de datos db4o

| Acción | Objetivo |
|---|---|
| Db4oEmbedded.openFile(<br>Db4oEmbedded.newConfiguration(),<br>"alumnos.db4o"<br>) | El método openfile abre una base de datos y si no existe la crea. Recibe dos parámetros, uno de tipo Configuration el cual se puede obtener realizando la llamada Db4oEmbedded.newConfiguration().<br>La interfaz *Configuration* contiene métodos para poder configurar db4o.<br>El segundo parámetro es el nombre del fichero donde se va a alojar la base de datos. |
| bd.close() | Se le insta al objeto de tipo *ObjectContainer* a cerrar la base de datos. |

# 9.5.2 ALMACENAR OBJETOS

```
public static void almacenarAlumnos(ObjectContainer bd)
  {
   alumno a1 = new alumno("Juan Gámez",23,8.75);
   bd.store(a1);
   System.out.println(a1.getNombre()+" Almacenado");
   alumno a2 = new alumno("Emilio Anaya",24,6.25);
   bd.store(a2);
   System.out.println(a2.getNombre()+" Almacenado");
   alumno a3 = new alumno("Ángeles Blanco",26,7);
   bd.store(a3);
   System.out.println(a3.getNombre()+" Almacenado");
}
```

# 9.5.3 RECUPERAR OBJETOS DE LA BASE DE DATOS (I)

```
public static void
    mostrarResultado(ObjectSet  res){
    System.out.println("Recuperados
    "+res.size()+" Objetos");
    while(res.hasNext()) {
        System.out.println(res.next());
    }
}
```

# 9.5.3 RECUPERAR OBJETOS DE LA BASE DE DATOS (II)

```
public static void
    muestraAlumnos(ObjectContainer bd)
    {
    alumno a = new alumno(null, 0, 0);
    ObjectSet res =
    bd.queryByExample(a);
    mostrarResultado(res);
}
```

# 9.5.3 RECUPERAR OBJETOS DE LA BASE DE DATOS (III)

```
public static void
    muestraAlumnos26(ObjectContainer
    bd) {
    alumno a = new alumno(null, 26, 0);
    ObjectSet res =
    bd.queryByExample(a);
    mostrarResultado(res);
}
```

# 9.5.4 ACTUALIZAR OBJETOS EN LA BASE DE DATOS

```
public static void
    actualizarNotaAlumno(ObjectContainer
    bd,String nombre,double nota) {
    ObjectSet res = bd.queryByExample(new
    alumno(nombre,0,0));
    alumno a = (alumno)res.next();
    a.setNota(nota);
    bd.store(a);
    muestraAlumnos(bd);
}
```

# 9.5.5 BORRAR OBJETOS DE LA BASE DE DATOS

```
public static void
  borrarAlumnoporNombre(ObjectContainer
  bd,String nombre) {
  ObjectSet res = bd.queryByExample(new
  alumno(nombre,0,0));
  alumno a = (alumno)res.next();
  bd.delete(a);
  muestraAlumnos(bd);
}
```


Ra-Ma®
Juan Carlos Moreno

# 9.6 CONSULTANDO LA BASE DE DATOS




# Tipos de consultas

- **Query By Example (QBE)**. Es el sistema ya visto anteriormente.
- **Native Queries (NQ)**. Son consultas nativas. Es la interfaz principal de la base de datos y aconsejado por los desarrolladores de db4o.
- **SODA (Simple Object Data Access)**. Es la API interna. Es mucho más potente que las dos anteriores y mucho más rápida dado que los dos tipos de consultas anteriores (QBE y NQ) tienen que ser traducidas a SODA para ejecutarse.

# Limitaciones al utilizar QBE

- Al contrario que con otros lenguajes de consulta no se pueden realizar expresiones avanzadas (por ejemplo operadores AND, OR, NOT, etc.).
- Hay que proporcionar un ejemplo con las limitaciones que ello conlleva.
- No se puede preguntar por objetos cuyo valor de un campo numérico sea 0, Strings vacíos o algún campo que sea nulo (*null*).
- Se necesita un constructor para crear objetos con campos no inicializados.

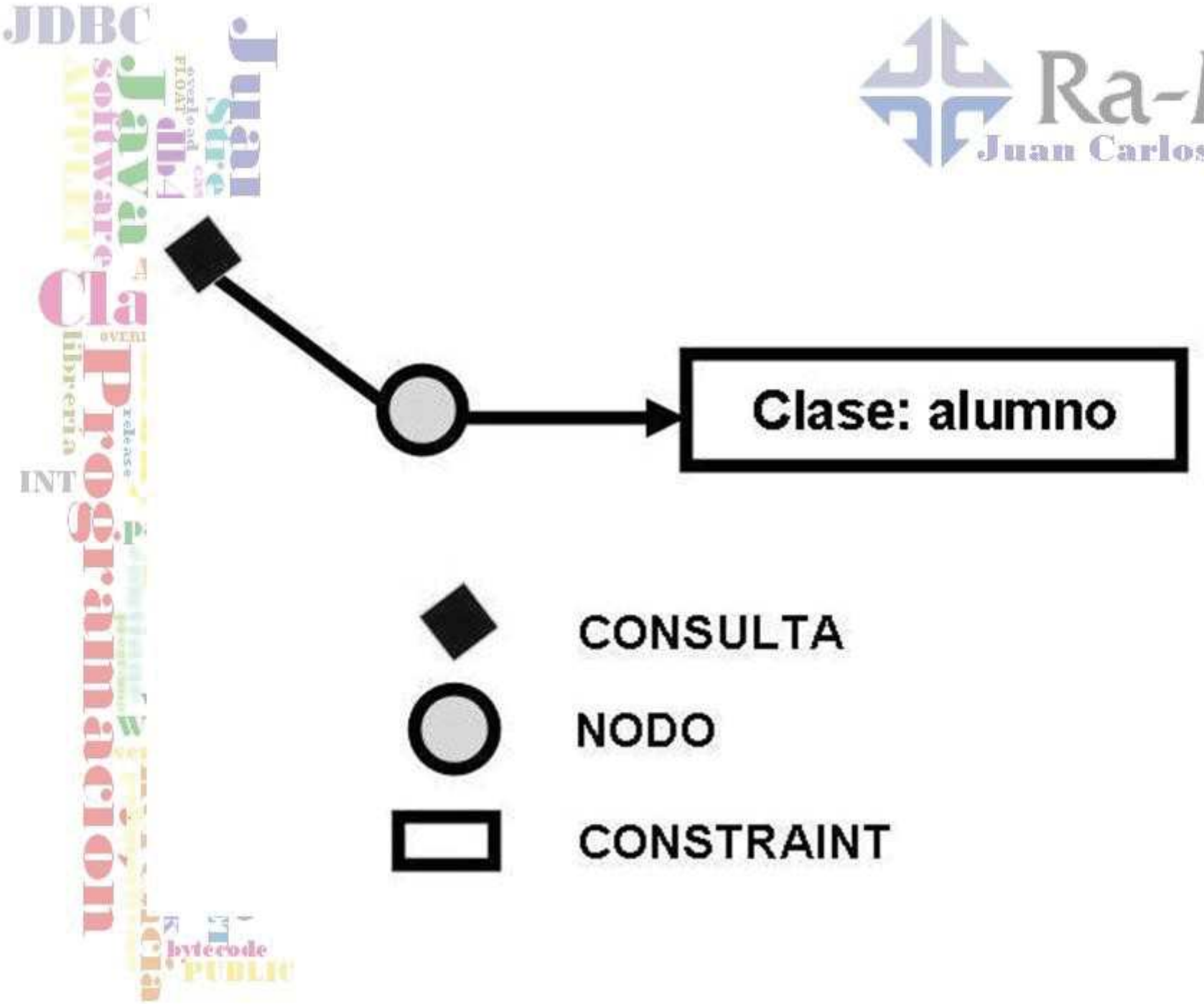
# 9.6.1 LIBRERÍA API SODA

- SODA = *Simple Object Database Access*.
- Esta librería presenta la ventaja que las consultas se ejecutan de la manera más rápida y permite generar una consulta dinámica.
- Para crear una consulta expresada en SODA se necesita un objeto de tipo *Query*
- Una vez creado este objeto *Query* se le van añadiendo *Constraints* (restricciones) para ir modelando el resultado que se quiere obtener.
- Una vez que ya se tiene modelada la consulta se ejecuta la consulta llamando al método execute() del objeto tipo *Query*.

# Ejemplo: listado de todos los objetos alumno

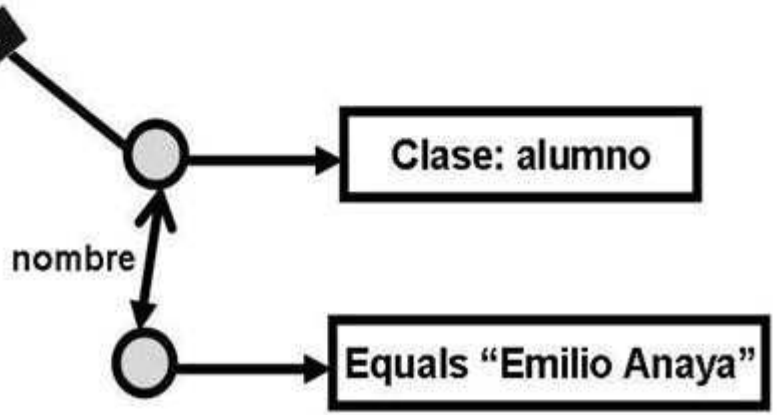


```
public static void
    consultaSODAAlumnos(ObjectContainer
    bd) {
    Query query=bd.query();
    query.constrain(alumno.class);
    ObjectSet result=query.execute();
    mostrarResultado(result);
}
```



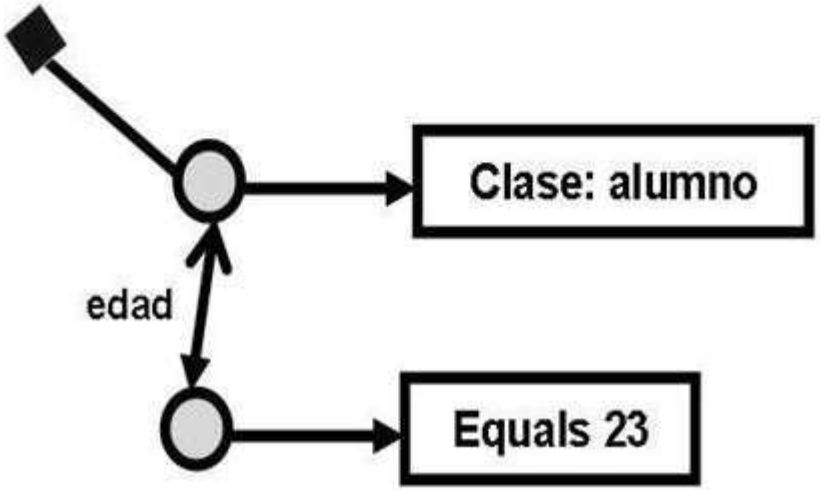
# Ejemplo 2: Alumno Emilio Anaya

```
public static void
  consultaSODAEmilio(ObjectContainer bd) {
  Query query=bd.query();
  query.constrain(alumno.class);
  query.descend("nombre").constrain("Emilio
  Anaya");
  ObjectSet result=query.execute();
  mostrarResultado(result);
}
```



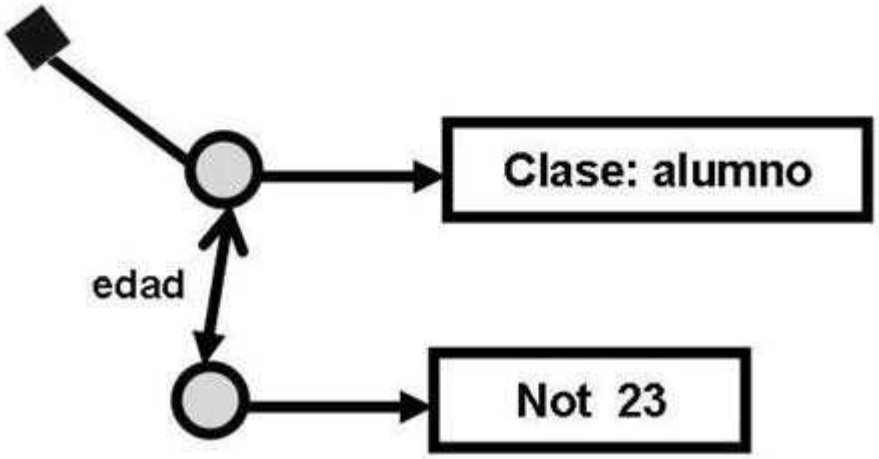
# Ejemplo 3: Alumnos con 23 años

```
public static void
  consultaSODAAlumVtres(ObjectContainer bd) {
  Query query=bd.query();
  query.constrain(alumno.class);
  query.descend("edad").constrain(23);
  ObjectSet result=query.execute();
  mostrarResultado(result);
}
```



# Ejemplo 3: Alumnos que no tienen 23 años

```
Query query=bd.query();
query.constrain(alumno.class);
query.descend("edad").constrain(23).not();
ObjectSet result=query.execute();
mostrarResultado(result);
```



# Ejemplo 3: Alumnos mayores de 23 años y menores de 25.

```
Query query=bd.query();
query.constrain(alumno.class);
Constraint
constr=query.descend("edad").constrain(25).smaller();
query.descend("edad").constrain(23).greater().and(con
str);
ObjectSet result=query.execute();
mostrarResultado(result);
```