**Mantenimiento de la persistencia de los objetos**

**Secuenciación**

Como ya sabemos, se define la persistencia de objetos al hecho de guardar toda la información que contiene un objeto manteniendo toda su estructura, dentro de una memoria secundaria (pendrive, DVD,etc) Para ello hay que usar o bien ficheros o bien bases de datos. Los ficheros se han cubierto en la unidad 7, en esta unidad se tratará como hacerlo usando bases de datos.

**Base de datos orientada a objetos**

Una base de datos orientada a objeto (BDOO) se caracteriza porque la información que guarda - como su propio nombre indica - son objetos, conservando toda su estructura de atributos y métodos.

Sn embargo, en una base de datos relacional había que extraer los datos que contenía un objeto para poder guardarlos en las diferentes tablas o al revés.

Gracias al uso de las bases de datos orientadas a objetos, es fácil guardar los datos que se manejan en un programa orientado a objetos, ya que conservan toda su estructura. De igual modo, a la hora de recuperarlos se recuperan en forma de objeto, permitiendo con ello hacer uso de los métodos de la clase a la que pertenece.

**Características de las bases de datos orientadas a objetos**

* Guarda la información de los objetos, conservando toda la estructura de la clase a la que pertenece
* En estas bases de datos, lo que identifica a cada objeto guardado en dicha base no son sus datos ni el nombre del objeto sino un identificador que se le da a cada objeto, el cual es único para cada objeto. Esto es lo que se conoce como OID (ObjectsIdentifier) Dicho identificador es independiente de los datos que contiene.
* En una misma base de datos se pueden almacenar objetos de diferentes tipos de clases (Empleados, Alumnos, Profesores…)
* Permite guardar la información de objetos estructurados, es decir, objetos donde alguno de los datos es de tipo objeto. Este tipo de base de datos permite manejar esta información de forma muy sencilla

**Instalación del gestor de bases de datos**

Hay muchos gestores de bases de datos para objetos (ODBMS, ObjectDatabase Management System). El que usaremos nosotros es el db4o. Sus siglas significan “Database 4(for) Objects”, lo cual significa base de datos para objetos

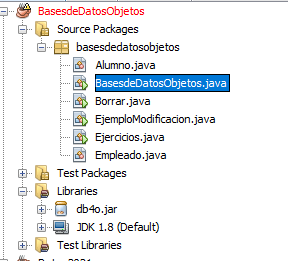
Db4o permite almacenar objetos Java de forma directa y fácil.

Algunas razones para elegir este gestor:

* De código abierto y libre bajo la GPL. Es la principal clave del éxito de este gestor de base de datos
* Hace que sea fácil la persistencia de objetos
* Es capaz de almacenar cualquier tipo de objeto, aunque la clase a la que pertenezca sea muy compleja. Aunque la clase sea muy compleja, el gestor es capaz de guardar la información de los objetos definidos con este tipo de clase con una sola línea de código
* Tiene un alto rendimiento
* La base de datos que crea db4o es un único fichero de extensión “.yap” (En las últimas versiones, ya no hay tal extensión)
* Algo muy importante de este gestor es que para poder trabajar con dicho programa no hay que instalar nada. Sólo hay que añadir a nuestro proyecto un fichero jar que contiene dicho gestor

Se tiene al lado de este documento el gestor (Le hemos cambiado el nombre a db4o.jar para abreviar)

Para usar dicho gestor lo que tenemos que hacer es simplemente agregar el fichero anteriormente indicado a nuestro proyecto a través de la biblioteca. Así, a partir de este momento, podremos usar todas las clases que tiene la librería, que permitirán gestionar una base de datos orientada a objetos.

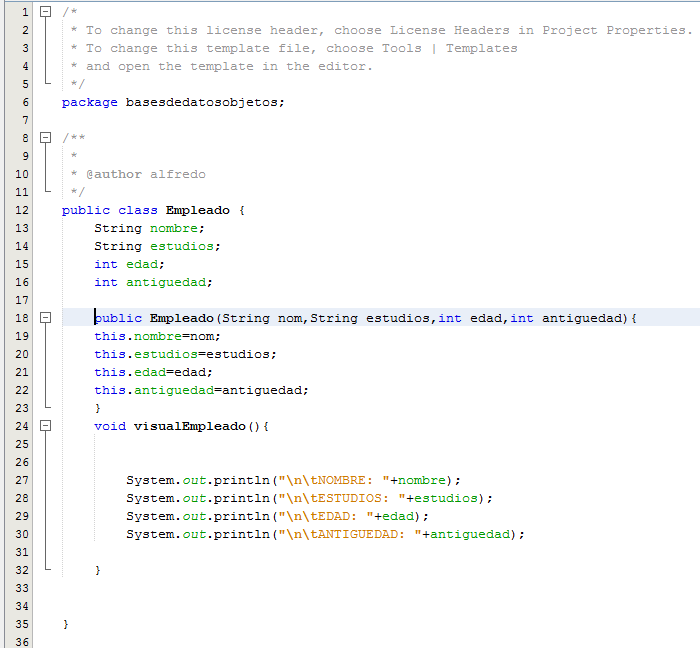


Muchos de los ejemplos que vamos a estudiar a lo largo de la unidad van a hacer uso de las siguientes dos clases:

*Clase Alumno*



*Clase Empleado*



**Creación de bases de datos**

La única instrucción necesaria para crear una base de datos orientada a objetos es ésta:

**ObjectContainerbaseDatos = Db4oEmbedded.openFile(“nombreBaseDatos”);**

**Db4oEmbedded** es una clase que pertenece a la librería del gestor db4o que hemos importado. Esa librería tiene un método **openFile** y dicho método permite crear y abrir una base de datos. Si la base de datos ya estuviese creada, se limitaría a abrirla. Exactamente, lo que hace este fichero es crear un fichero (si no existiera) con el nombre indicado y con extensión “.yap”; dicho fichero lo guarda en la raíz del proyecto. Este fichero representa la base de datos.

El método **OpenFile()**devuelve un objeto de tipo **ObjectContainer;** este objeto es el que va a permitir el manejo de la base de datos, debido a que dicha clase tiene métodos para poder llevar a cabo operaciones como insertar, borrar, consultar,…objetos de una base de datos

Por ello, una vez abierta la base de datos, podremos insertar objetos. Para ello hay que utilizar el método **store** de la clase **ObjectContainer.** Dicho método requiere un parámetro que es el objeto que queremos guardar en la base de datos.

**Ejemplo**

En este ejemplo, vamos a crear una base de datos, asignándole el nombre de “BDAlumEmp”



Vemos que la base de datos contiene objetos de diferente tipo: de tipo Empleado y de tipo Alumno

Es muy importante cerrar la base de datos cuando sepamos que ya no la vamos a usar. La instrucción para cerrar la base de datos es

**Nombre\_objeto\_ObjectContainer.close();**

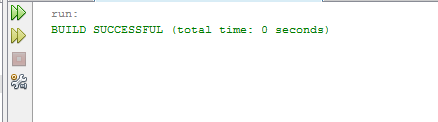
En nuestro ejemplo sería baseDatos.close();

Si no la cierra el programador, el programa mostrará un mensaje como:

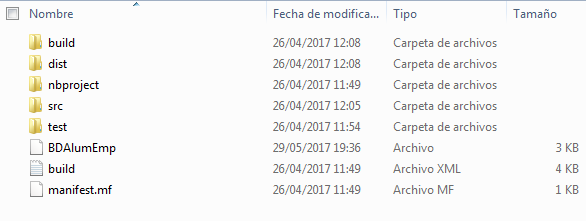
[db4o 8.0.276.16795 2018-01-29 15:14:50] nombreBaseDatosclosedbyShutdownHook

Muchos de los ejemplos y ejercicios siguientes los vamos a hacer usando esta base de datos recién creada. Debe recordarse que esta base de datos es un fichero con el nombre de que se le ha dado a dicha base de datos con extensión .yap y está guardado en la raíz del proyecto.

El resultado de ejecución del anterior programa es el siguiente:



Es decir, no nos devuelve errores pero tampoco sabemos si ha guardado bien -más allá de que el archivo está creado



Y no lo sabemos porque en realidad no le hemos dicho al ordenador que nos muestre nada.

Para que el ordenador nos muestre elementos de objetos, debemos utilizar distintos mecanismos de consulta

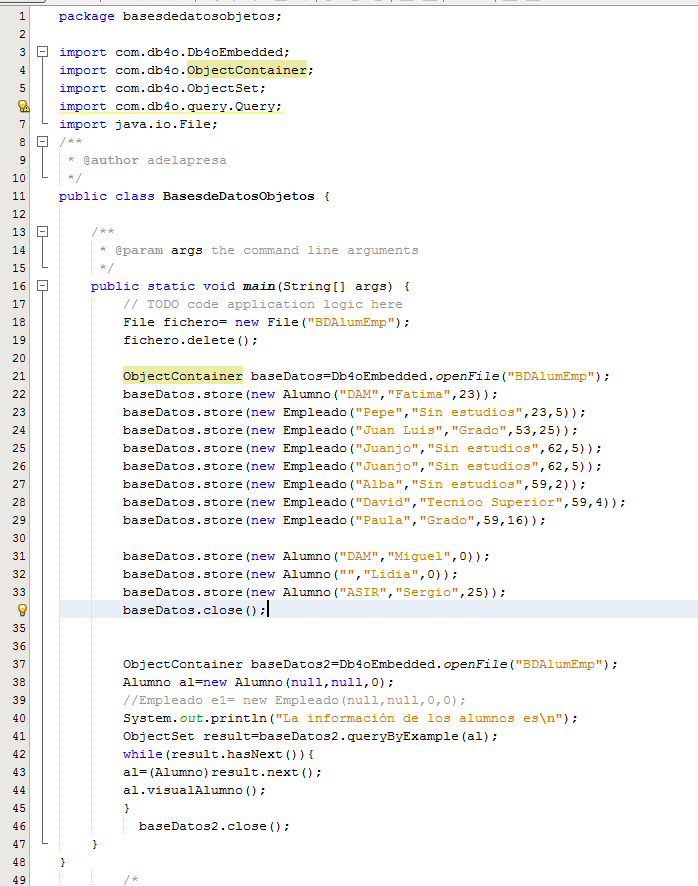
**Mecanismos de consulta**

**QBE**

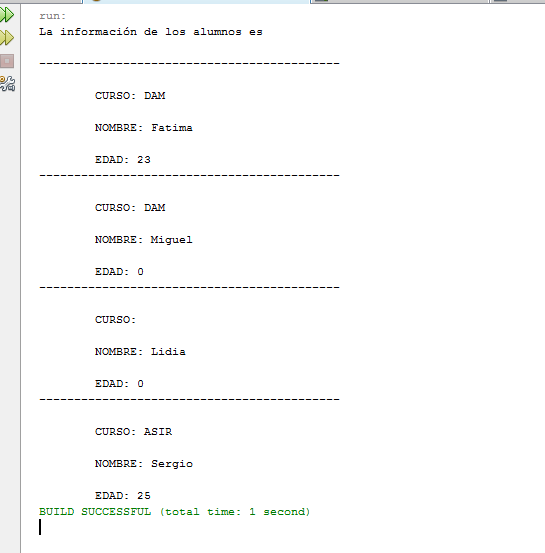
Las consultas sencillas se hacen por QBE. QBE viene de QueryByExample, que significa “Consulta por ejemplo”

Su nombre se debe a la forma que tiene de recuperar los objetos que contiene la base de datos. Para ello utiliza el método **QueryByExample()** que tiene la clase **ObjectContaine**r. En la llamada a dicho método se le enviará un objeto que servirá de ejemplo con el fin de indicar cuáles son los objetos que queremos recuperar de la base de datos; los objetos que recupere serán aquellos que sean del mismo tipo que el objeto que recibe y que tengan los mismos datos. Si en algún dato no queremos plantear ninguna condición, a dicho objeto ejemplo se le asignará un valor **null**si es un atributo de tipo referencia o el valor 0 si es un atributo de tipo primitivo.

Ejemplo 1. El siguiente código visualiza todos los objetos pertenecientes a la clase Alumno de la Base de Datos

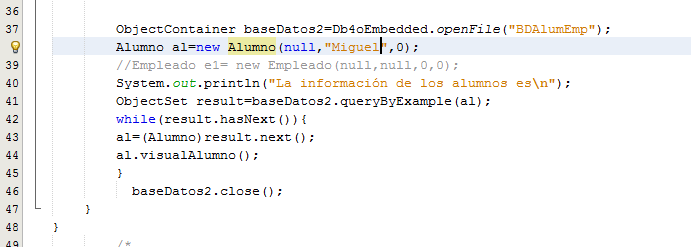


En ejecución nos produce la siguiente salida:

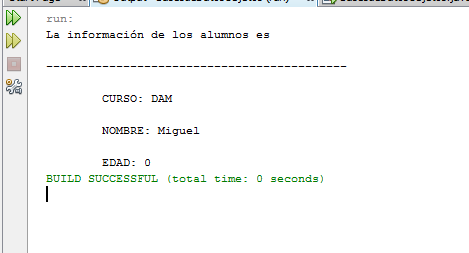


Es decir, devuelve todos los objetos de tipo Alumno que tenga la base de datos, sin hacer ninguna selección entre ellos; esto se debe a que no se plantea ninguna condición sobre sus datos ya que el objeto ejemplo tiene valor 0 (para los datos de tipo numérico) y valor null (para los datos de tipo referencia)

Ahora vamos a recuperar todos los objetos de la clase Alumno que se llaman “Miguel”



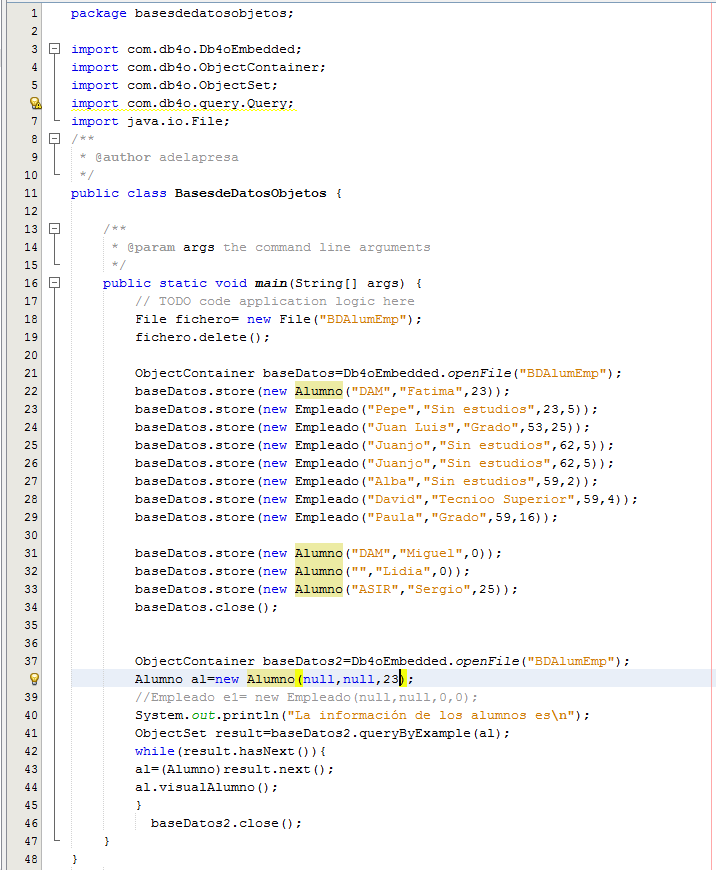
Que nos produce como salida lo siguiente:



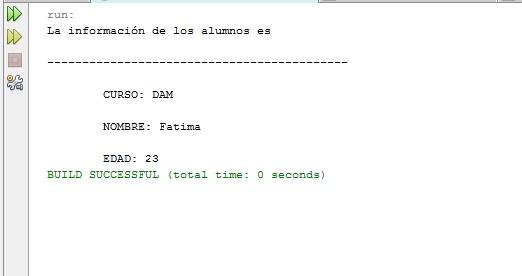
El principal inconveniente de este tipo de consultas es que no se permite seleccionar los objetos que contenga la base de datos, estableciendo como condición que uno de los datos numéricos del objeto sea un cero o un dato de tipo referencia que contenga el valor null.

Ejercicio

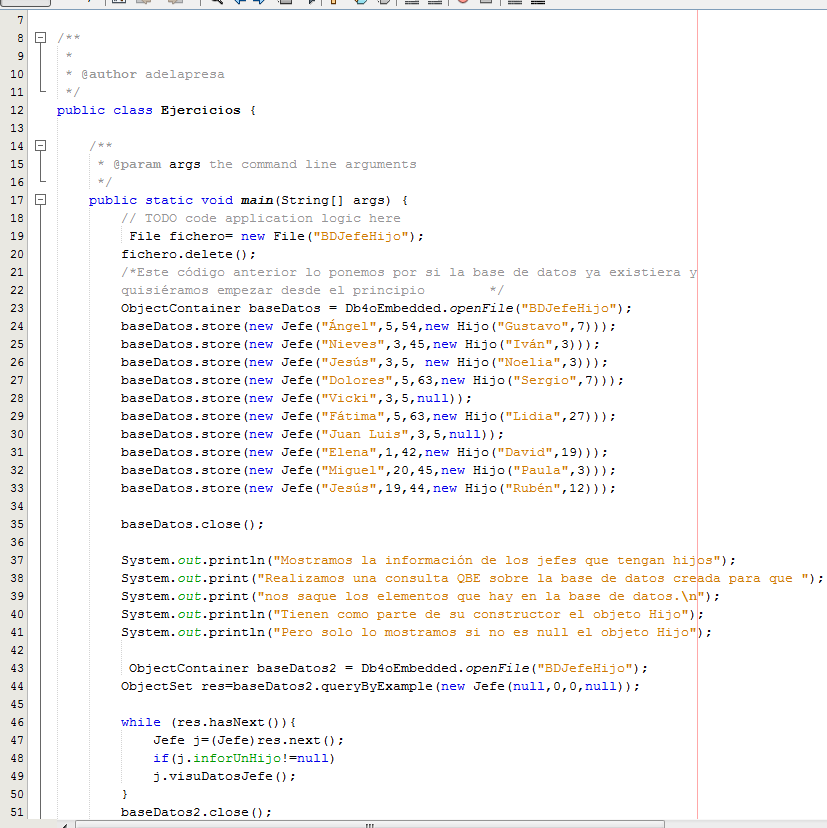
Escribe un programa que visualice la información de todos los alumnos que haya en la base de datos que tengan 23 años



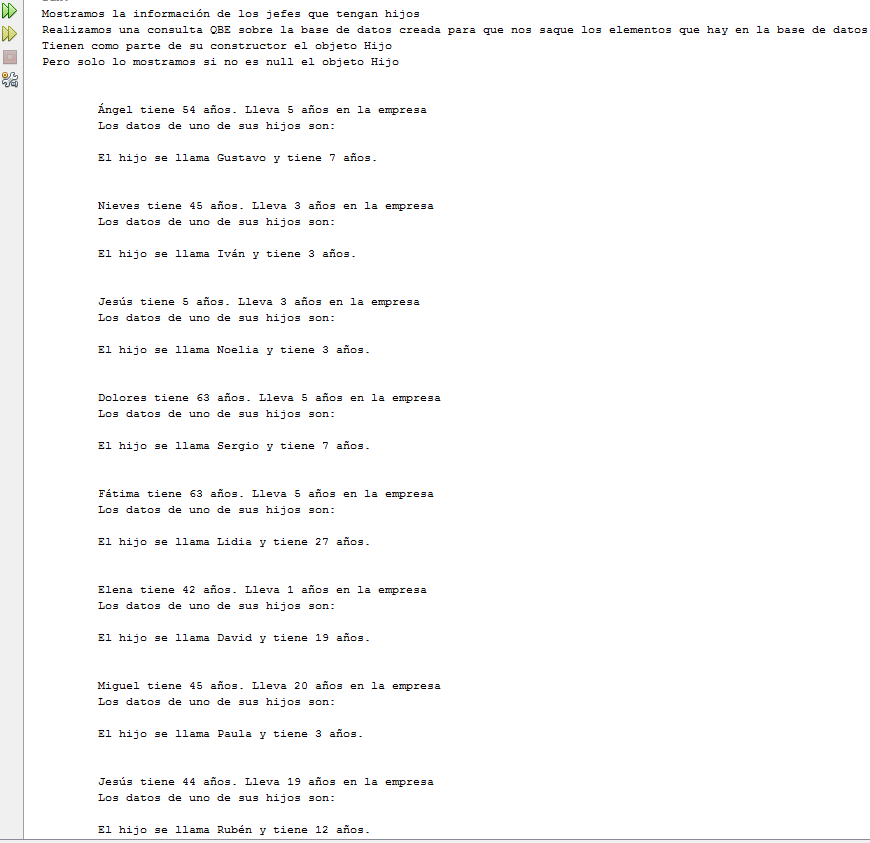
La ejecución nos da:a



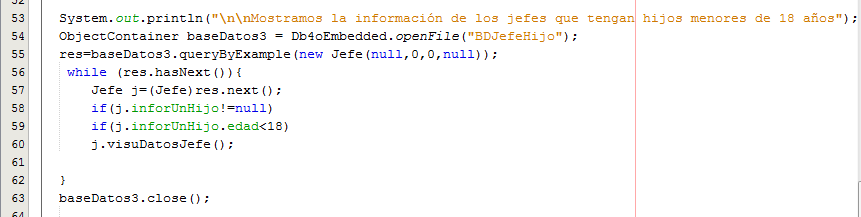
Ejercicio 1:



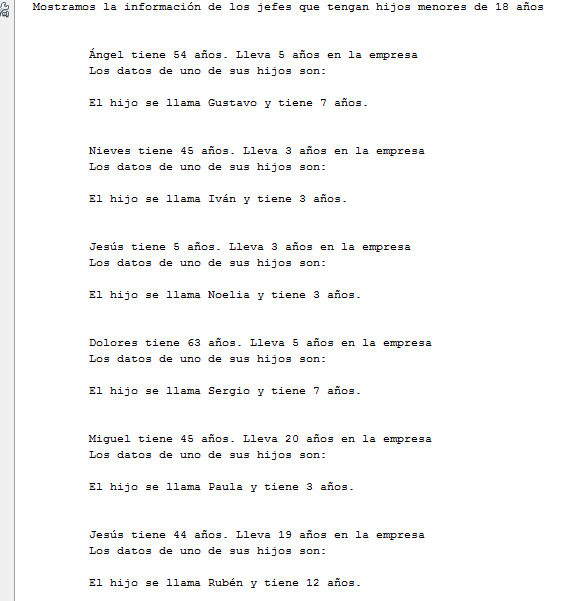
En ejecución se tiene:



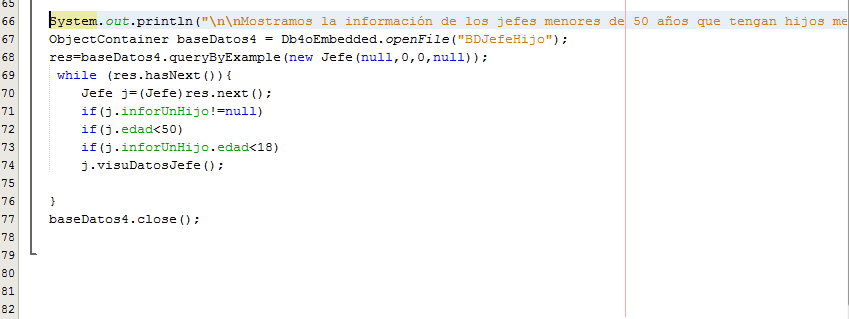
Ejercicio 2



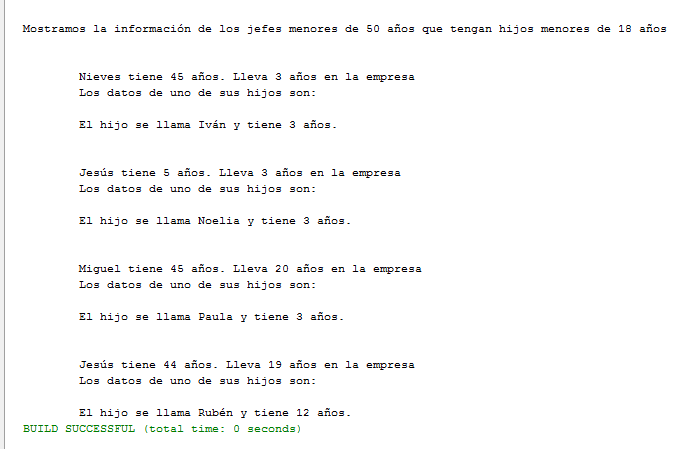
Este código nos genera la siguiente salida:



Ejercicio 3.



Este código nos produce la siguiente salida en ejecución:



**SODA**

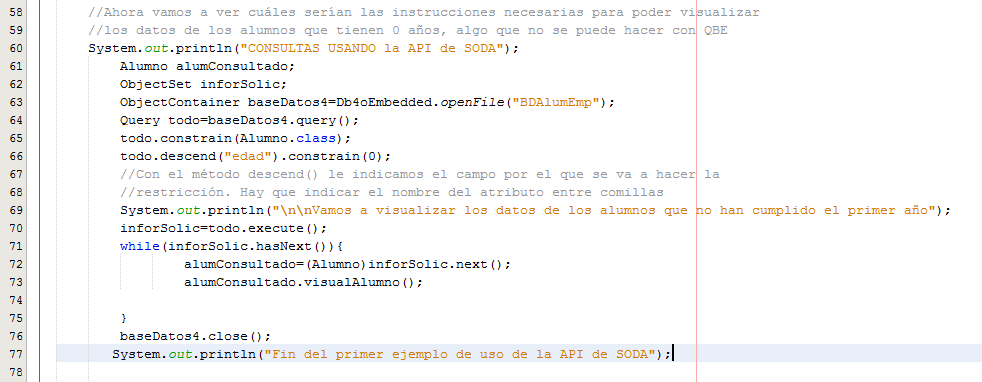
Cuando hay que realizar consultas más complicadas se hace uso de unas determinadas clases que están en una librería llamada SODA. Una de las clases que tiene este API es la clase **Query**. Vamos a ir explicando su uso

**El lenguaje de consultas: sintaxis, expresiones, operadores**

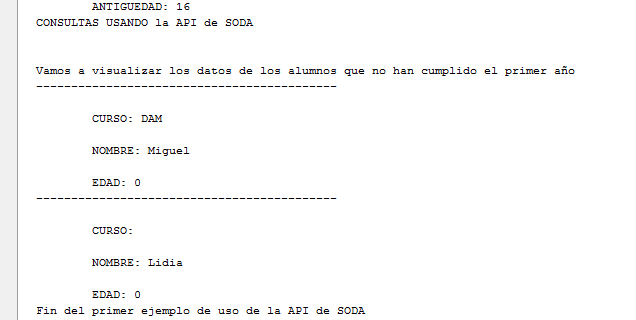
Estas consultas las haremos empleando el API de SODA.

Usando el API de SODA pueden realizarse consultas más complejas que con QBE

Una consulta sencilla, usando dicha API podría ser la siguiente:



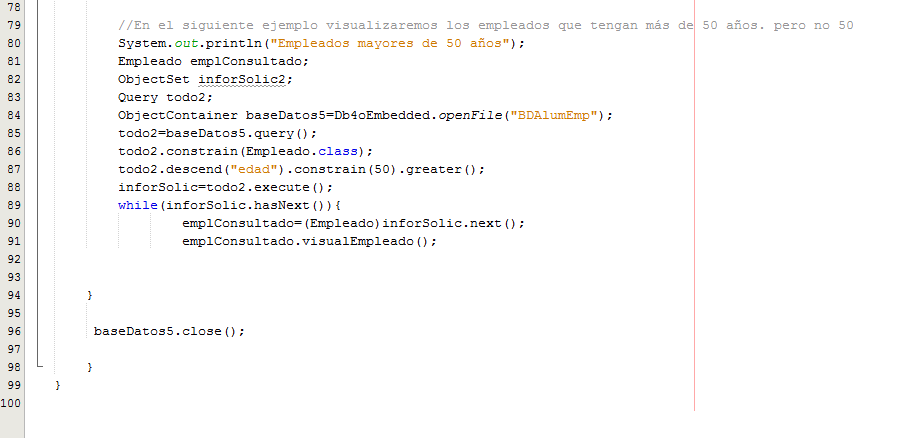
Que nos produce lo siguiente cuando se ejecuta:



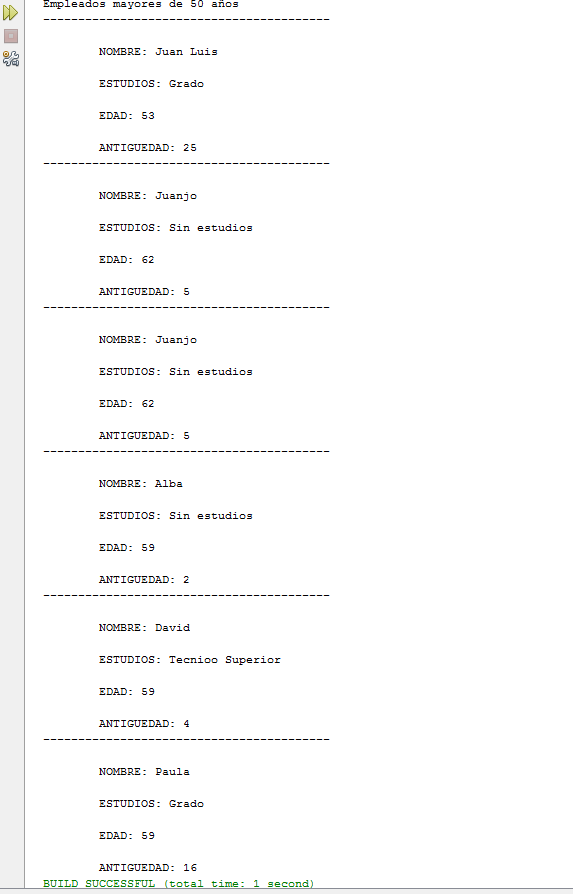
Vemos que se usa el método query() de la clase **ObjectContainer.** Dicho método recoge toda la información que contiene la base de datos y lo guarda en un objeto de tipo **Query.** Dicho objeto tiene el método **constrain()** con el fin de poder seleccionar la información deseada.

Posteriormente, el método **execute()**, vuelca toda la información que tenga el objeto **Query** en un objeto de tipo **ObjectSet,** ya que es dicha clase la que tiene una serie de métodos que van a permitir poder manejar la información que contiene.

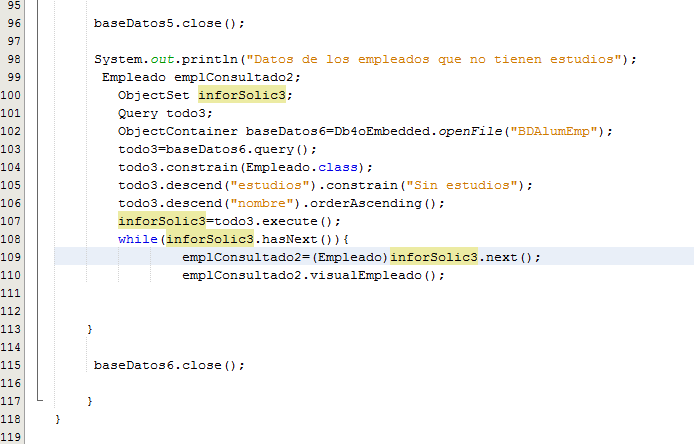
Otra consulta, un poco más compleja, visualizará a los empleados que tengan más de 50 años, pero no 50



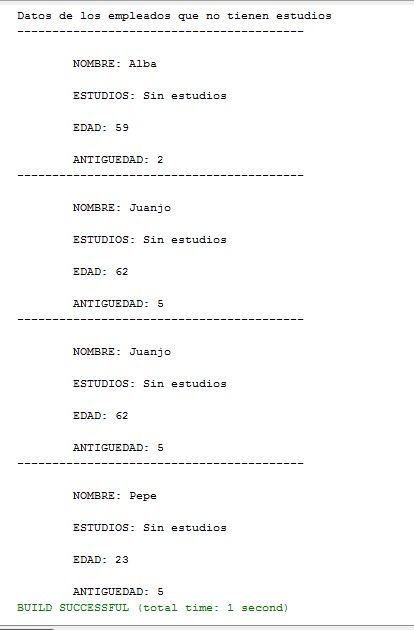
La ejecución nos produce el siguiente resultado:



En el siguiente ejemplo se visualizarán los empleados que no tengan estudios. Se van a mostrar los datos ordenados de forma descendente por el nombre

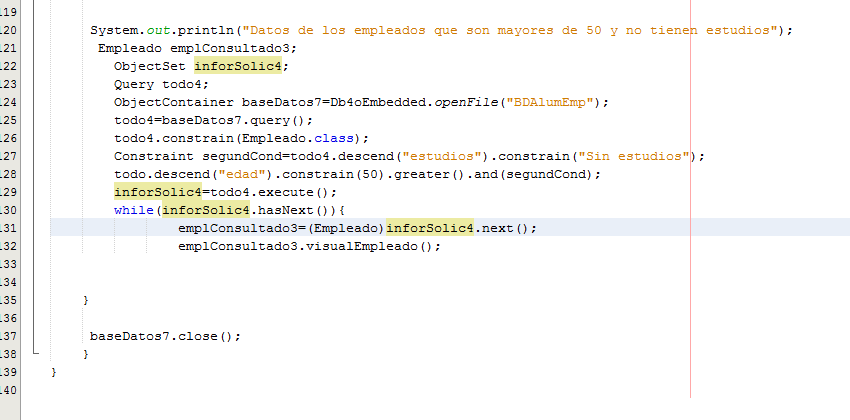


La ejecución de este trozo de código nos da:

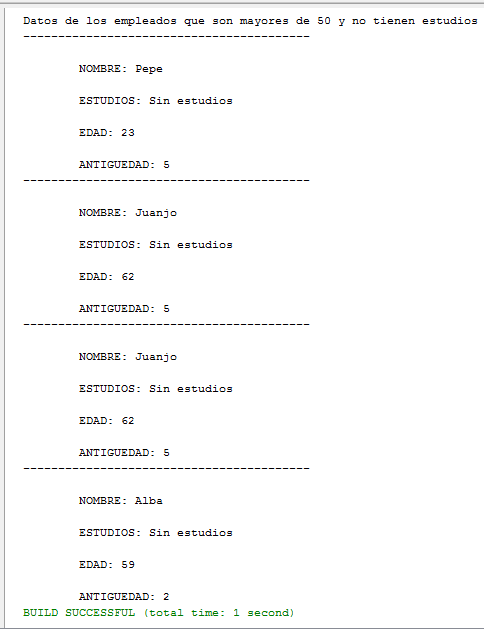


En el siguiente ejemplo visualizaremos los empleados que tienen más de 50 años y no tienen estudios

El código es el siguiente:



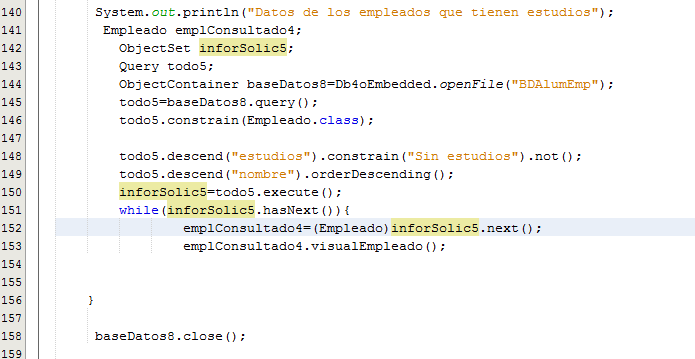
La ejecución del anterior programa produce:



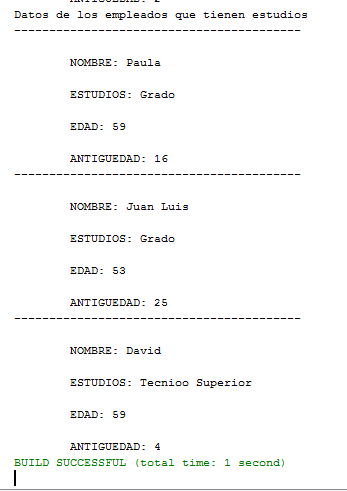
**Ejercicio**

Realiza un programa que visualice los datos de todos los empleados, excepto de aquellos que no tengan estudios. Visualízalos de forma ascendente por el nombre

El código es el siguiente:



La ejecución de este código nos produce el resultado siguiente:



**Recuperación, modificación y borrado de la información**

**Modificación**

Para actualizar un objeto primero lo tenemos que recuperar de la base de datos para conocer cuál es el identificador de dicho objeto. Luego modificamos sus datos y lo volvemos a insertar en la base de datos.

Para poder entender esto, vamos a ver estos dos ejemplos:

Las siguientes instrucciones:

baseDatos2.store(al1);

baseDatos2.store(al1);

No guardan el objeto dos veces, sino que el primero sustituye al segundo, ya que tienen el mismo identificador:

Y si hacemos:

Alumno al1= new Alumno();

Alumno al2= new Alumno();

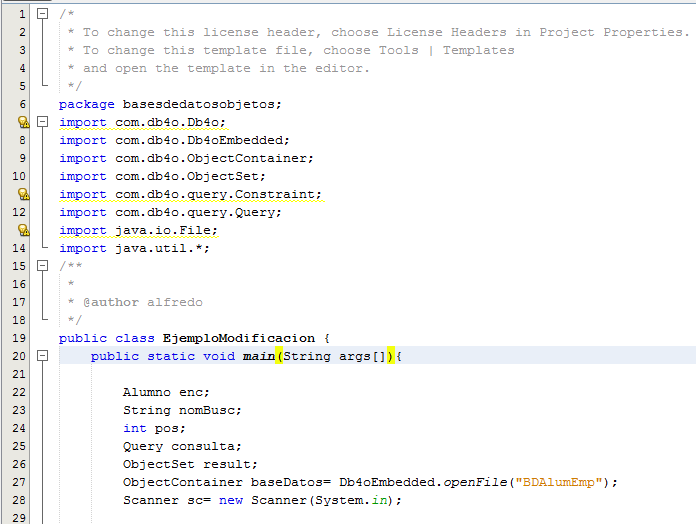
al1 = al2;

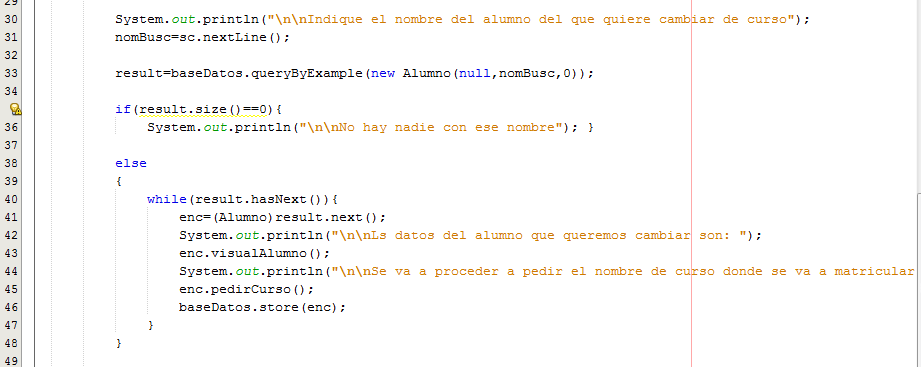
baseDatos2.store(all);

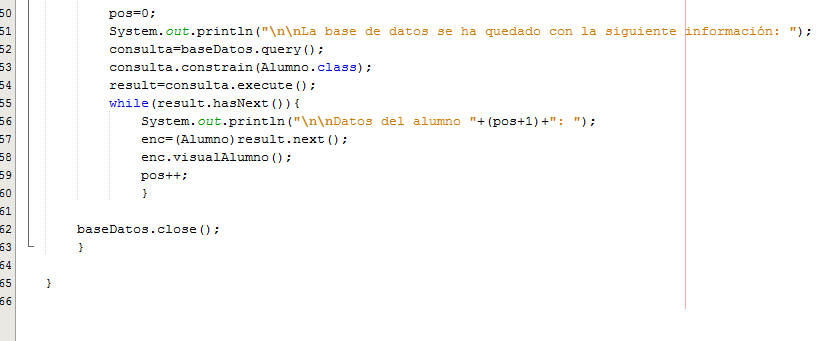
Insertará un solo objeto que es al2, ya que al1 tiene el mismo identificador que al2.

En el siguiente ejemplo, vamos a estudiar cómo modificar el curso donde está matriculado un determinado alumno cuyo nombre coincida con el insertado a través del teclado.

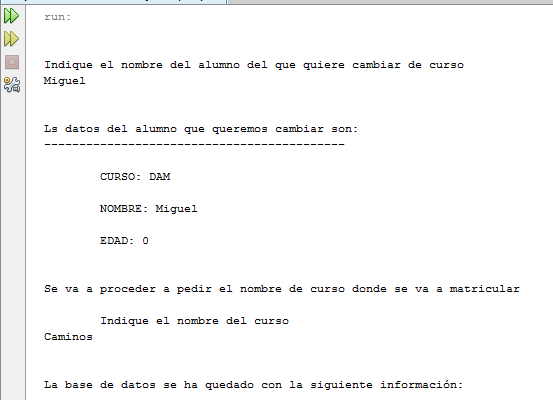
El código es:

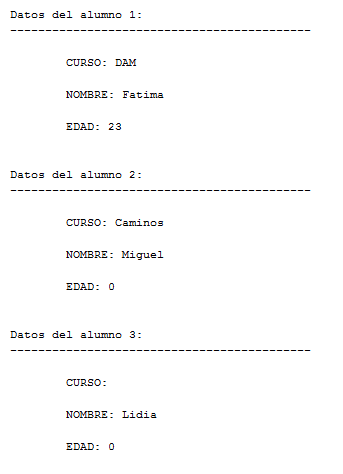






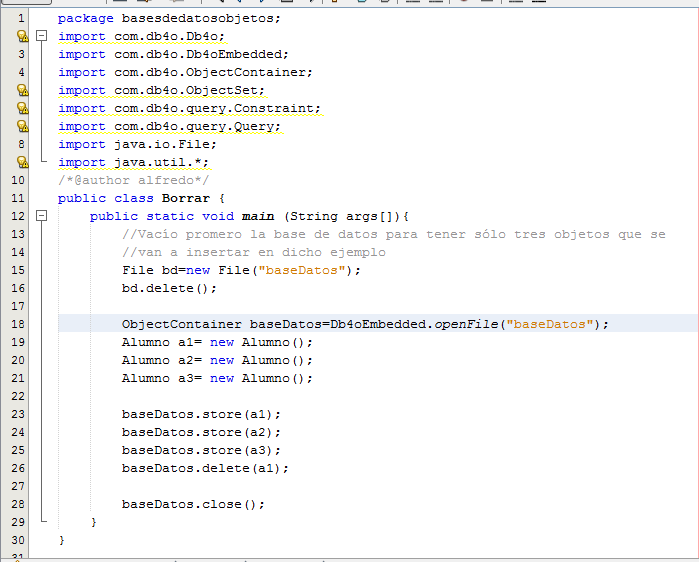
La ejecución del programa produce:



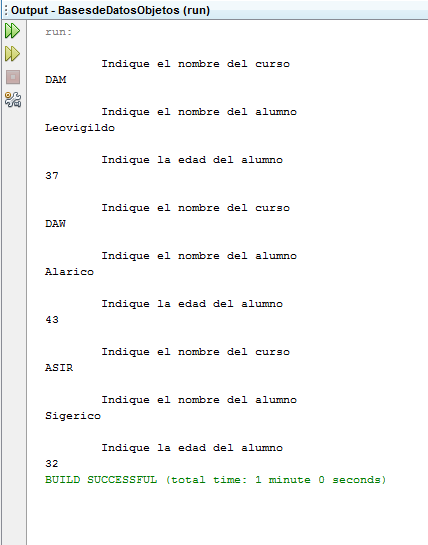


**Borrar**

Para borrar objetos en una base de datos hay que usar el método **delete** de la clase **ObjectContainer**



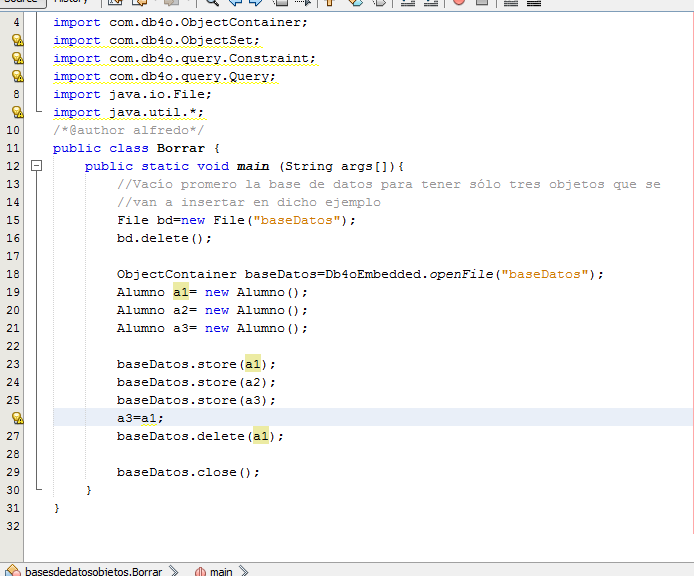
En ejecución este código produce lo siguiente:



El objeto que borra es aquel cuyo identificador coincida con el del objeto que se envía como parámetro.

Si no hay ningún objeto en la base de datos con dicho identificador, no se borrará ningún objeto.

Así, en el siguiente ejemplo, lo que haría el programa sería borrar el objeto de la base de datos que tenga el identificador del objeto a1, ya que a3 contiene el identificador de dicho objeto

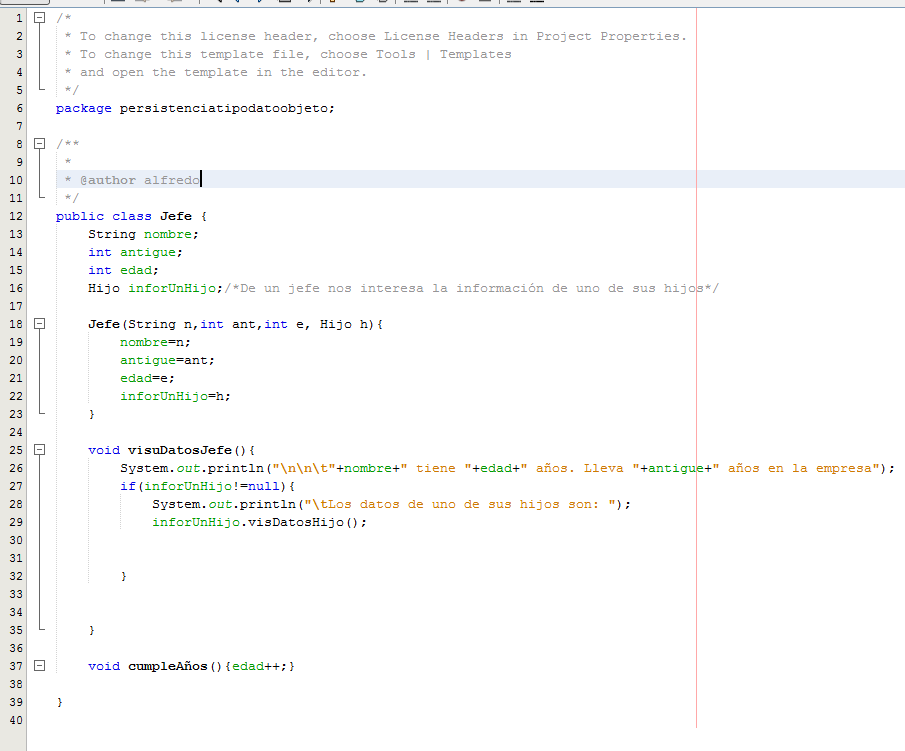


**Tipos de datos objeto. Atributos y métodos**

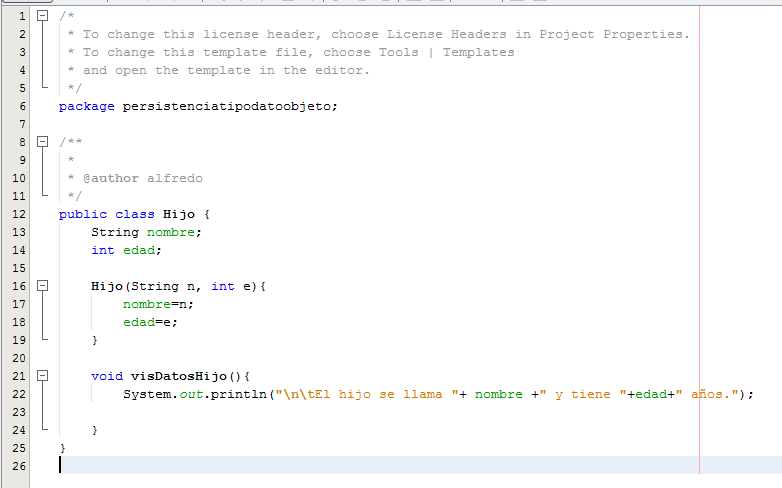
A partir de aquí van a estudiarse cómo se manejan ciertos objetos donde alguno de sus datos son de tipo referencia, es decir, objeto.

Vamos a estudiar cómo se realizan las distintas operaciones sobre este tipo de objeto. Para ello, vamos a hacer uso de las siguientes clases:

Clase Jefe



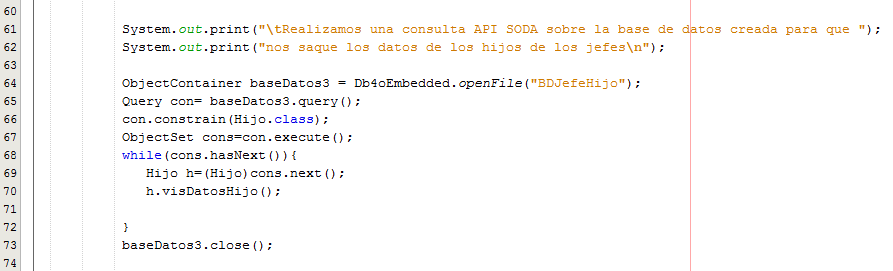
Clase Hijo



El código para crear una nueva base de datos que contenga este tipo de objetos es el siguiente (Se muestra hasta el momento que cerramos la base de datos tras hacer la consulta después de haber insertado los elementos, ya que utilizaremos esta clase principal para realizar más consultas)

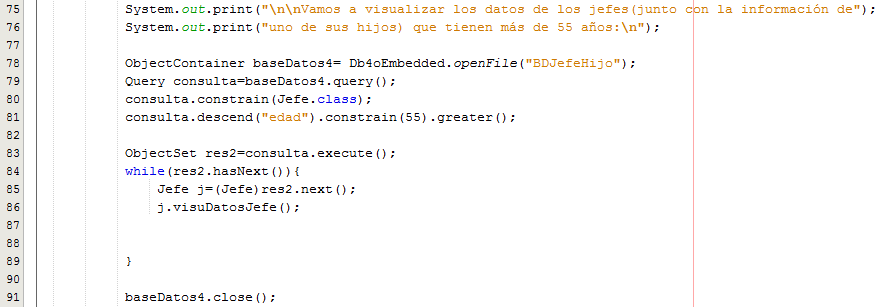


Nuestra siguiente consulta será la que nos saque los datos de los hijos de los jefes

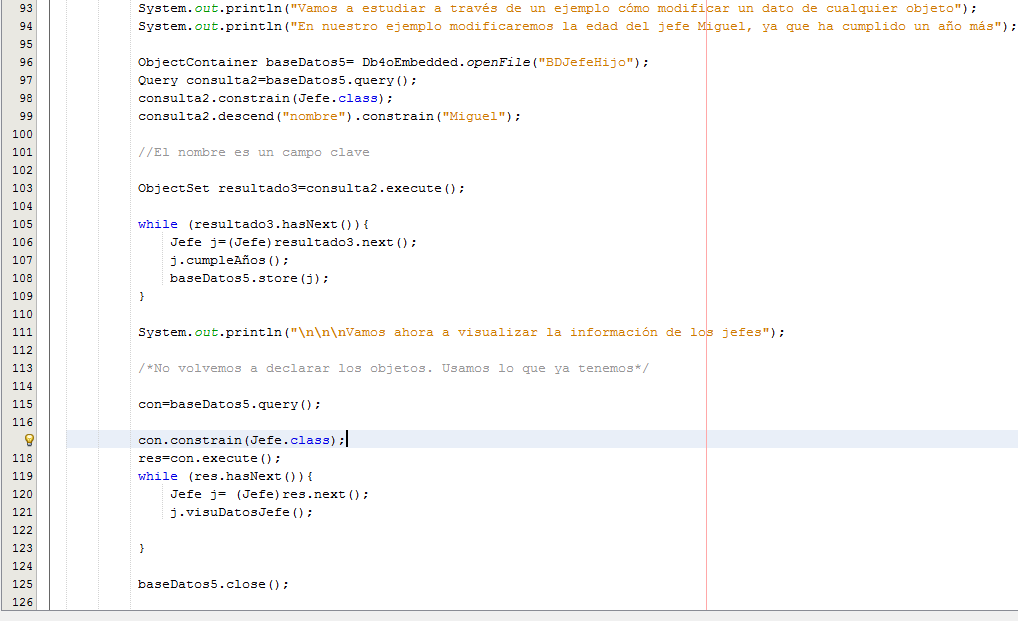


Ahora visualizamos los datos de los jefes -junto con la información de uno de sus hijos- que tienen más de 55 años

El código que resuelve este problema es el siguiente:



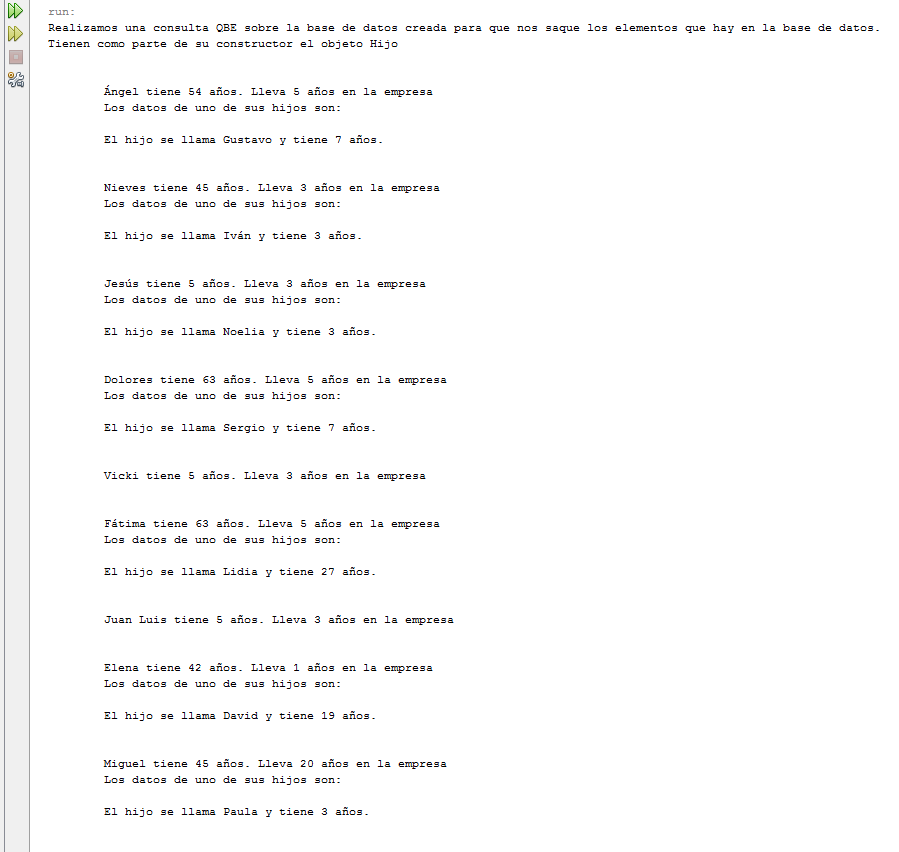
La siguiente consulta es la que nos permite añadirle un año más al Jefe cuyo nombre es “Miguel”.

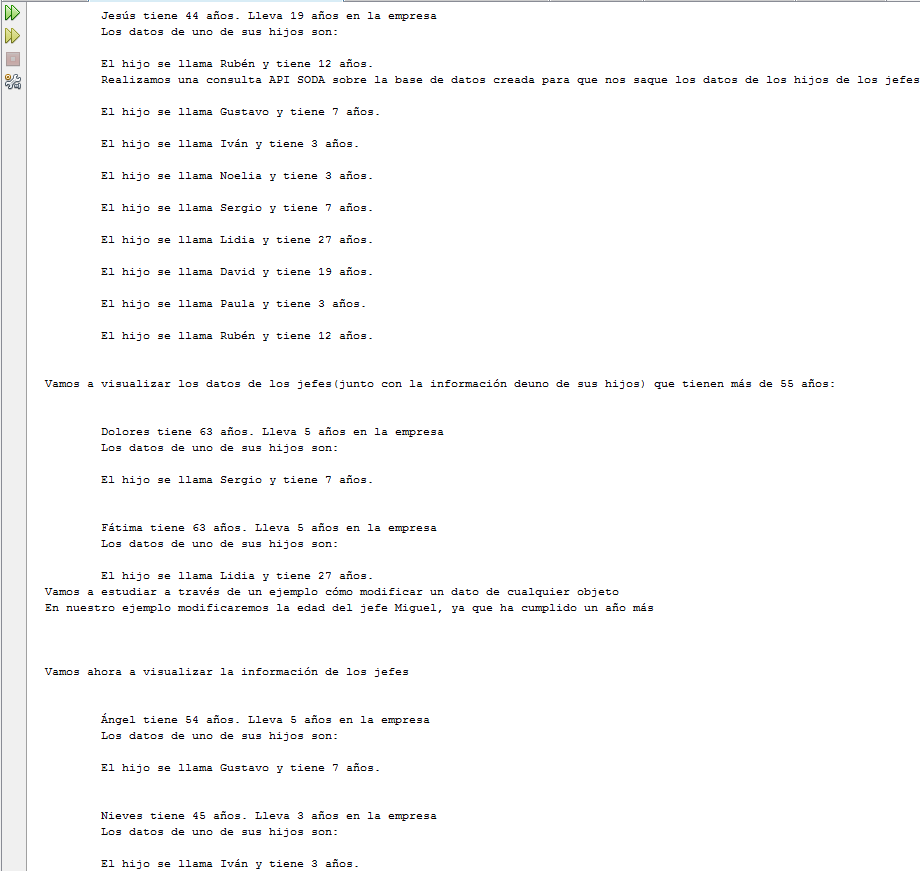


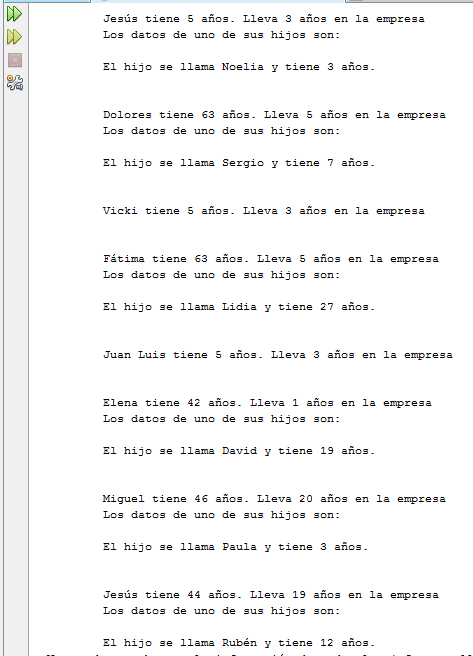
La siguiente consulta es la que nos permite borrar todos los jefes que lleven menos de diez años en la misma empresa.

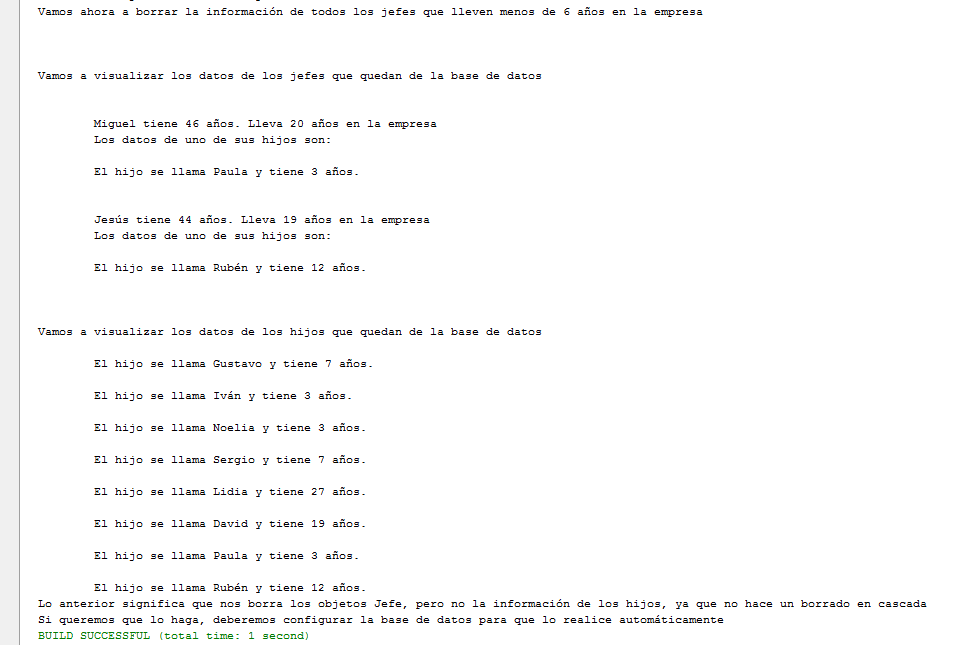


La ejecución completa de este código nos da lo siguiente:





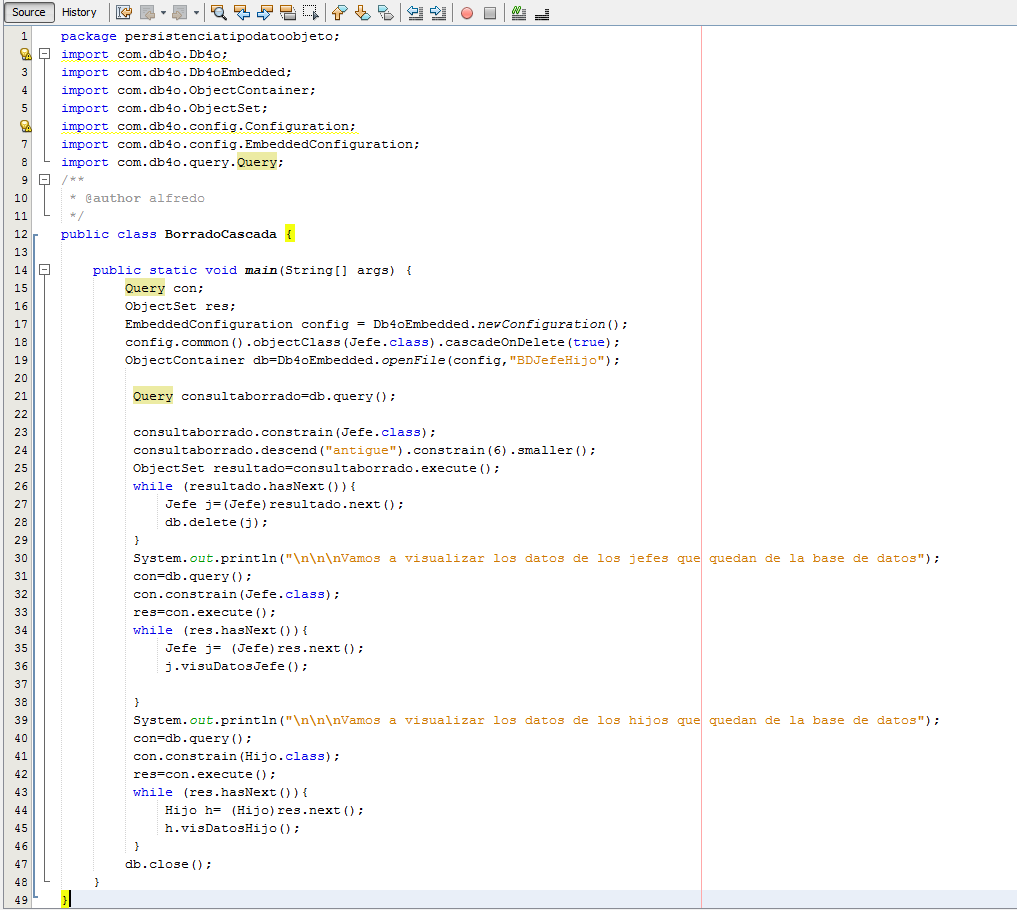




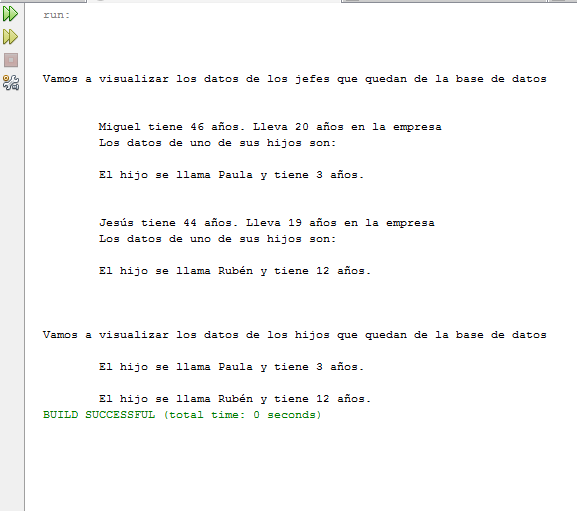
**Ejercicio**

Investiga cómo configurar la base de datos para que lleve a cabo un borrado en cascada. Para ello, hay que tener en cuenta que el método que lleva a cabo esta operación se llama: cascadeOnDelete. Pruébalo con el ejemplo anterior.

El código que resuelve el ejercicio es el siguiente:



Y la ejecución del mismo produce la siguiente salida:

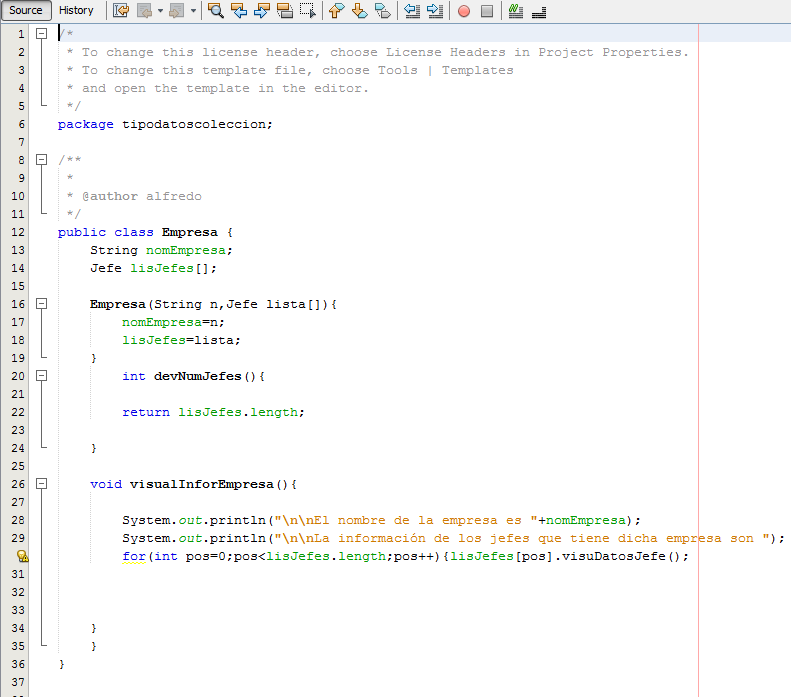


**Tipos de datos colección**

Llamamos colecciones a aquellos tipos como pilas, listas, colas, tablas. Todos estos tipos tienen en común que permiten almacenar más de un objeto

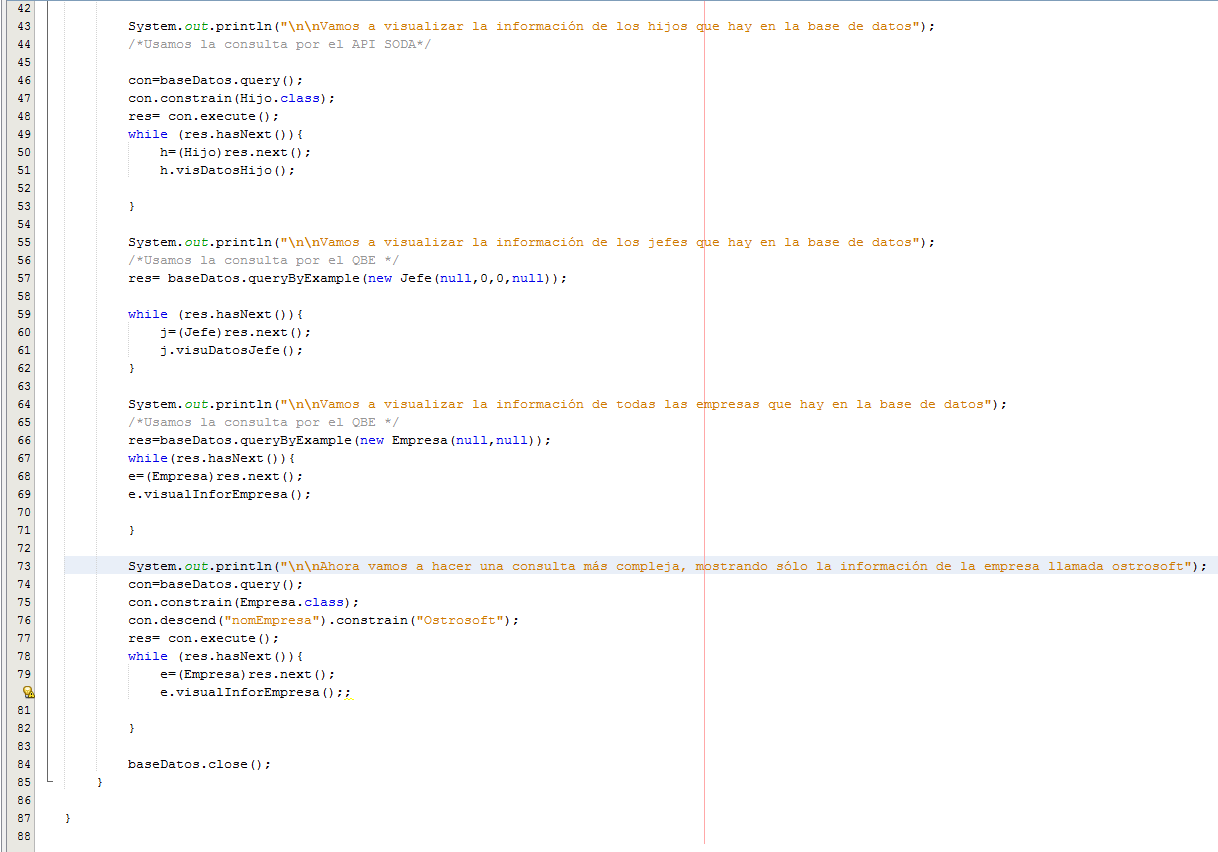
En el siguiente ejemplo se usarán tanto las clases Jefe e Hijo anteriores como la siguiente clase

Clase Empresa.



En el siguiente ejemplo de clase principal mostramos inserción y visualización y realización de una consulta más compleja.





Este programa en ejecución nos da:

