4.2: Persistencia de objetos: db4o

A. Goñi, J. Ibáñez, J. Iturrioz, J.A. Vadillo

OCW 2013

informatika fakultatea
facultad de informática

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

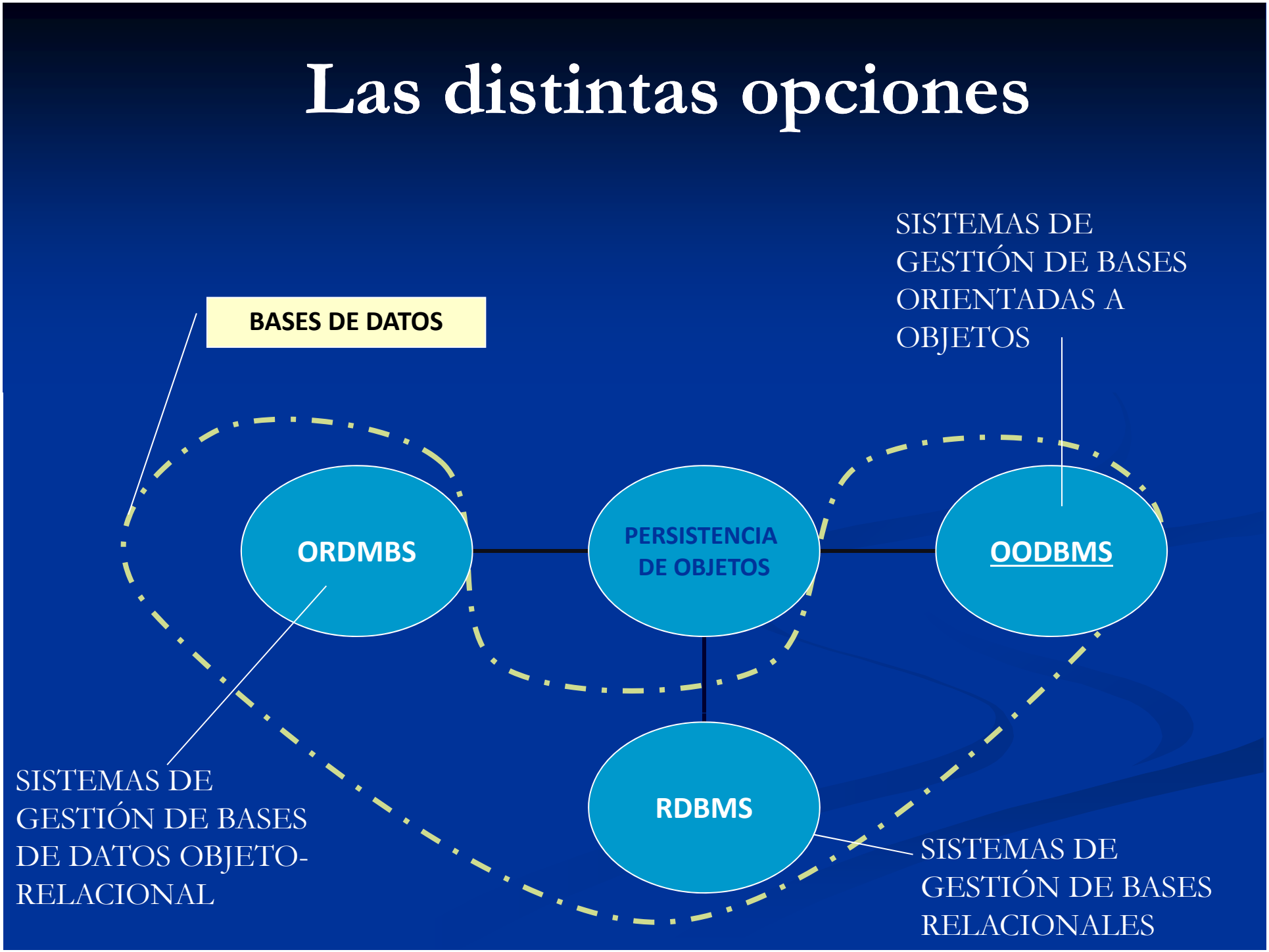
Índice

- Introducción
- Características de Db4o
- Creación de bases de datos
- Guardar objetos
- Consultar objetos
- Actualizar objetos
- Borrar objetos
- Herencia
- Transacciones
- Configuración

Introducción

- El almacenamiento y la recuperación de los objetos del dominio es una tarea importante en la programación
- Hay que asegurar que los objetos se almacenen en la memoria, esto es, hay que ocuparse de proporcionar la persistencia de los objetos.
- Hay distintas opciones para almacenar los objetos en un sistema orientado a objetos.

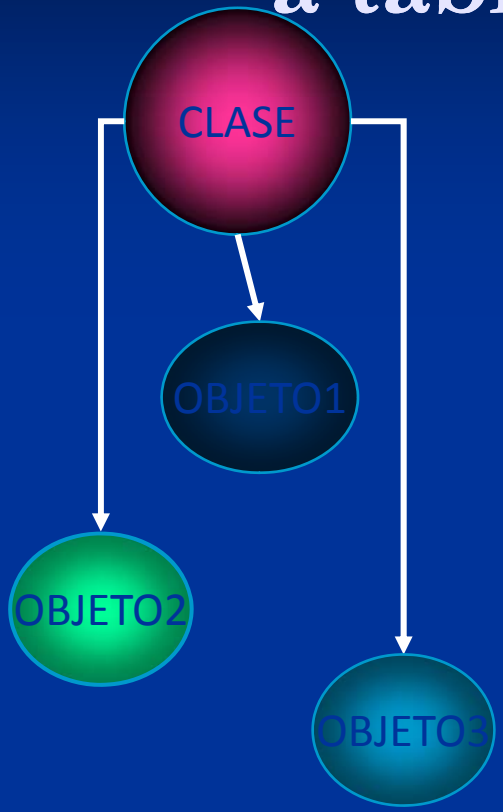
Las distintas opciones



Almacenamiento de Objetos en Sistemas de Gestión de Bases de Datos Relacionales

- Los objetos se guardan en bases de datos relacionales
- Los objetos se guardan en tablas:
  - Una tabla está formada por un conjunto de registros, filas o tuplas
  - Cada tupla está formada por atributos
- Hay que programar el almacenamiento de los objetos en tablas

De objetos del dominio a tablas relacionales



TRANSFORMACIÓN

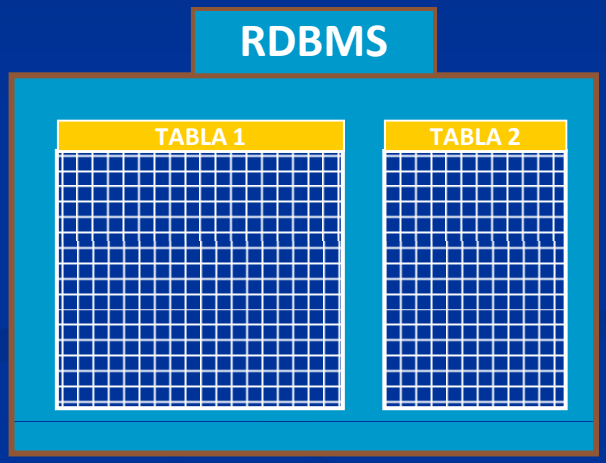
TUPLA 1

TUPLA 2

TUPLA 3

<ATRIBUTO1>

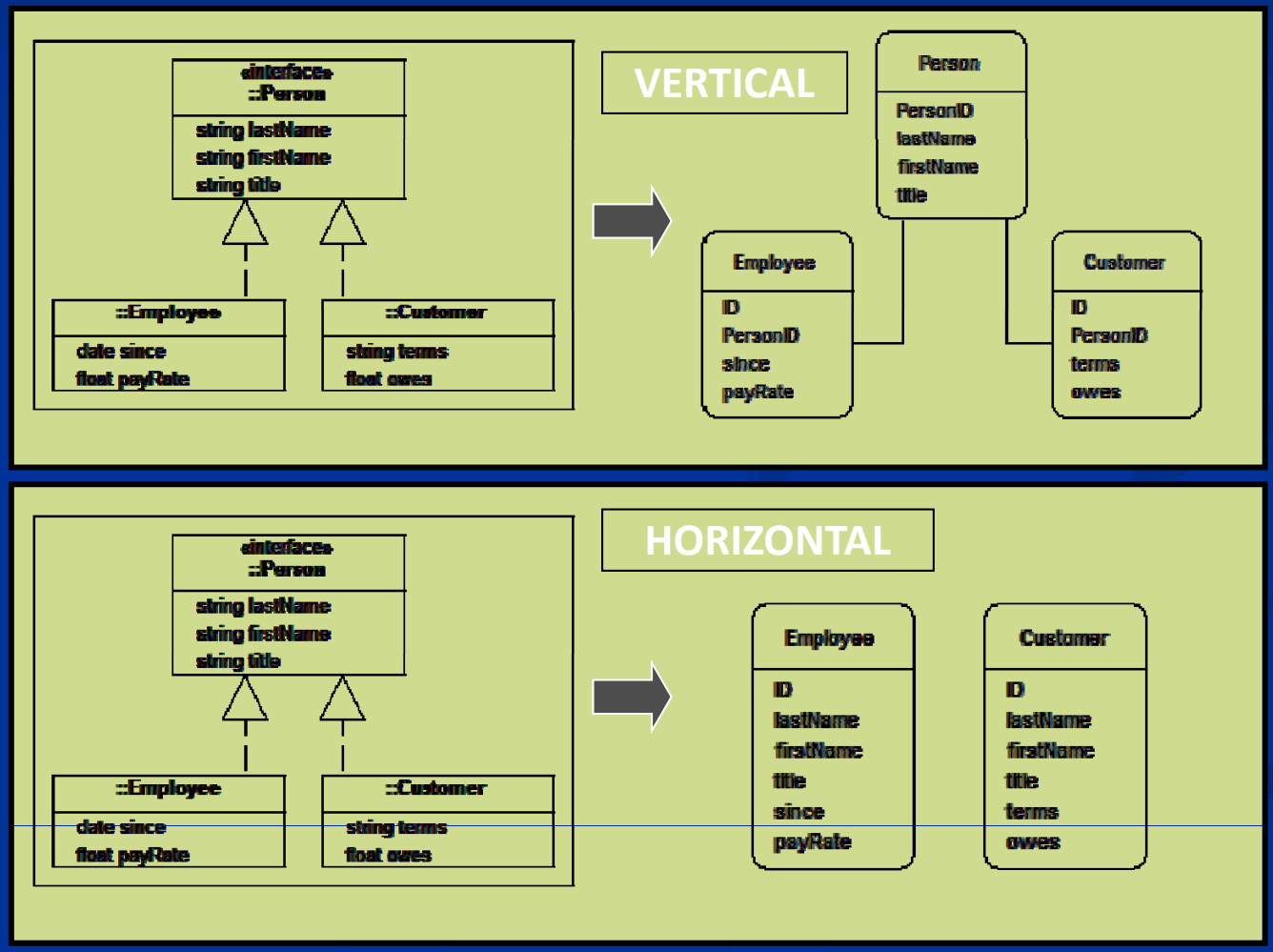
<ATRIBUTO N>



ENTORNO DE OBJETOS

ENTORNO DE DATOS

Hay distintas opciones



De objetos del dominio a objetos de BDOO

- Los objetos se almacenan directamente en bases de datos OO sin ninguna transformación (no hay tablas ni tuplas con atributos)
- Los BDOO nos proporcionan transparencia para almacenar los datos
- El esquema de clases (modelo del dominio) es el esquema de la BDOO
- Se facilita el trabajo de programación

De objetos del dominio a objetos de BDOO

Identificadores de los objetos

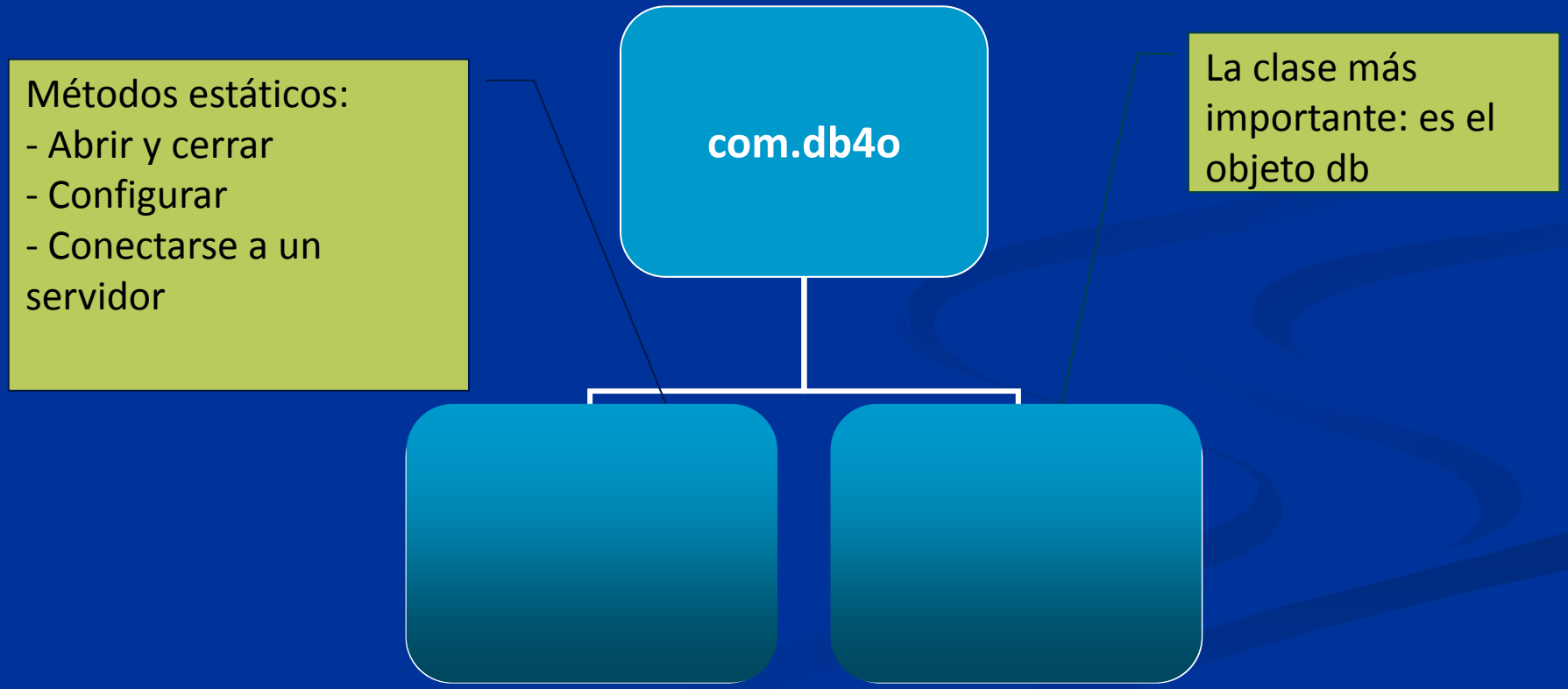
- OID's : identificadores de los objetos, que están almacenados dentro de cada objeto, y que permiten navegar entre objetos relacionados.
- Los OIDs no son visibles para los usuarios y los programadores.
- El identificador de un objeto no cambia, aunque cambie los valores de sus atributos.

Características de Db4o

- Db4o es un SGBDOO nativo
- Desarrollado en Sillicon Valley
- Se puede integrar fácilmente dentro de una aplicación
- Puede utilizarse para aplicaciones simples (standalone) o aplicaciones distribuidas (Cliente/Servidor)
- Funciona en entornos Java y .NET

Creación de un base de datos db4o

- Con las clases com.db4o.Db4o y com.db4o.ObjectContainer



Abrir y cerrar una base de datos

.openFile(String file)
.close()

```
ObjectContainer db = Db4o.openFile("Archivo.yap"); // Abrir la base de datos


try {
        // Trabajar con la base de datos
finally {


   db.close();   // Antes de salir, cerrar la base de datos
```

Objeto que contiene la base de datos

Ejemplo: La clase Pilot

```
public class Pilot {
  private String name;
  private int points;
  public Pilot(String name,int points) {
    this.name=name;
    this.points=points;
  }
  public int getPoints() {
    return points;
  }
  public void addPoints(int points) {
    this.points+=points;
  }
  public String getName() {
    return name;
  }
  public String toString() {
    return name+"/"+points;
  }
}
```

Crear/Guardar objetos

.store(Object o)

```
Pilot pilot1 = new Pilot("Michael Schumacher",100);

db.store(pilot1);           // db.set(pilot1)

System.out.println("Stored "+pilot1);
```

OUTPUT:

Stored Michael Schumacher/100

Crear otro objeto

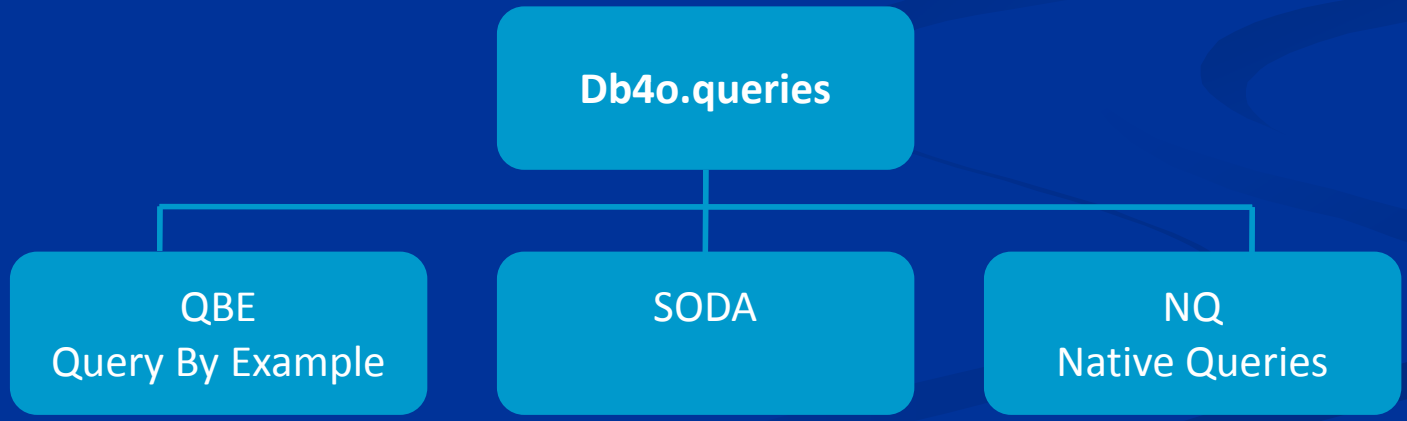
```
Pilot pilot2 = new Pilot(”Rubens Barrichelo",99);

db.store(pilot2);

System.out.println("Stored "+pilot2);
```

OUTPUT:

Stored Rubens Barrichelo/99

Preguntar por objetos en db4o

- 3 tipos de lenguajes de interrogación
  - Query by Example (QBE): utilizando un prototipo
  - Native Queries (NQ): las preguntas se hacen en el lenguaje de programación utilizado
  - Simple Object Database Access (SODA): definiendo preguntas dinámicas por medio de nodos



Quering by Example (QBE)

1. Preguntas sencillas
2. Se crea un objeto "prototipo" de pregunta con los valores de los atributos rellenos con los valores por los que se quiere preguntar
    1. El valor o y el null son valores "reservados" que significa que no se quiere restringir
3. Se usa un método .get al que se le pasa como parámetro un objeto "prototipo"
4. Se obtiene como respuesta un objeto ObjectSet
5. Restricciones
    - No se pueden realizar preguntas que contengan expresiones compuestas (AND, OR, NOT, etc.)
    - No se pueden preguntar por condiciones que contengan el 0 o el valor null
    - Se necesita un constructor de objetos

Ejemplo: Recuperar todos los objetos Pilot

.get(Object o)
.queryByExample(Object o)

```
Pilot proto = new Pilot(null,0);

ObjectSet result=db.get(proto);

listResult(result);
```

```
ObjectSet result=db.get(Pilot.class);
```

```
public static void listResult(ObjectSet result) {
  System.out.println(result.size());
    while(result.hasNext()) {
      System.out.println(result.next());
    }
}
```

OUTPUT:
2
Michael Schumacher/100
Rubens Barrichello/99

Ejemplo: Recuperar algunos objetos Pilot

- Los pilotos que tienen 100 puntos

```
Pilot proto = new Pilot(null,100);

ObjectSet result=db.get(proto);

listResult(result);
```

OUTPUT:
1
Michael Schumacher/100

Native Queries

- Se escriben en el mismo lenguaje que estamos utilizando
- Se comprueban en tiempo de compilación.
- Dentro de la consulta permiten llamar a los métodos de los objetos

Native Queries

1. Para realizar la consulta, hay que crear un objeto de una clase anónima que implementa la interfaz Predicate **[new Predicate()]**
2. La pregunta se define en el método
   - **boolean match(Object o)**
3. En la implementación del método match, se definen las condiciones de la pregunta: aquellos objetos que devuelven true son los que formarían parte de la respuesta
4. El método **ObjectSet query(Predicate p)** devuelve un objeto ObjectSet que contiene todos los objetos que cumplen el predicado p

Native Query

- Obtener los pilotos con 100 puntos

```
Predicate<Pilot> galdera=new Predicate<Pilot>(){
      public boolean match(Pilot pilot) {
       return pilot.getPoints() == 100;
      }
}

ObjectSet pilots = db.query(galdera);
```

Native Queries. Sort

Ordenar los pilotos de mayor a menor número de puntos

```
Comparator<Pilot> pilotCmp = new Comparator<Pilot>() {
    public int compare(Pilot p1, Pilot p2) {
        return (p1.getPoints()-p2.getPoints());
    }
};
```

Resultado del método compare:

if (a>b) return >0 (por ejemplo 1)

if (a==b) return 0

if (a<b) return <0 (por ejemplo -1)

Native Queries. Sort

Ahora la pregunta se realiza con el objeto comparator

```
Predicate<Pilot> galdera=new Predicate<Pilot>(){
        public boolean match(Pilot pilot) {
         return true;
       }
}
Comparator<Pilot> pilotCmp = new Comparator<Pilot>() {
    public int compare(Pilot p1, Pilot p2) {
       return (p1.getPoints()-p2.getPoints());
    }
}

ObjectSet pilots = db.query(galdera,pilotCmp);
```

Actualizar objetos db4o

```
ObjectSet result=db.get(new Pilot("Michael Schumacher",0));

Pilot found=(Pilot)result.next();

found.addPoints(11);
db.store(found);
System.out.println("Added 11 points for "+found);


retrieveAllPilots(db);
```

OUTPUT:

```
Added 11 points to Michael Schumacher/111
2
Michael Schumacher/111
Rubens Barrichello/99
```

Ejemplo: La clase Car

```
public class Car {
private String model;
private Pilot pilot;

public Car(String model) {
this.model=model;
this.pilot=null;}

public Pilot getPilot() {
return pilot;}

public void setPilot(Pilot pilot) {
this.pilot = pilot;}

public String getModel() {
return model; }

public String toString() {
return model+"["+pilot+"]";}
}
```

Actualizar objetos compuestos

```
// SESIÓN 1
ObjectSet result=db.query(new Predicate() {
public boolean match(Car car){
return car.getModel().equals("Ferrari"); }
});
Car found=(Car)result.next();
found.getPilot().addPoints(1);
db.store(found);
listResult(result);
```

```
OUTPUT:
1
Ferrari[Somebody else/1]
```

```
// SESIÓN 2
ObjectSet result=db.query(new Predicate() {
public boolean match(Car car){
return car.getModel().equals("Ferrari");}
});
listResult(result);
```

```
OUTPUT:
1
Ferrari[Somebody else/0]
```

ERROR: ¡ NO HA ACTUALIZADO EL NÚMERO DE PUNTOS!

Actualizar objetos compuestos

```
// CONFIGURAR LA BD PARA QUE AL ACTUALIZAR, ACTUALICE OBJETOS INCRUSTADOS
Db4o.configure().objectClass("Car").cascadeOnUpdate(true);
```

```
// SESIÓN 1
ObjectSet result=db.query(new Predicate() {
public boolean match(Car car){
return car.getModel().equals("Ferrari"); }
});
Car found=(Car)result.next();
found.getPilot().addPoints(1);
db.store(found);
listResult(result);
```

```
OUTPUT:
1
Ferrari[Somebody else/1]
```

```
// SESIÓN 2
ObjectSet result=db.query(new Predicate() {
public boolean match(Car car){
return car.getModel().equals("Ferrari");}
});
listResult(result);
```

```
OUTPUT:
1
Ferrari[Somebody else/1]
```

AHORA SÍ LO RECUPERA BIEN

Borrar objetos

.delete(Object o)

```
ObjectSet result=db.get(new Pilot("Michael Schumacher",0));
Pilot found=(Pilot)result.next();
db.delete(found);
System.out.println("Deleted "+found);

retrieveAllPilots(db);
```

Borrar el objeto

OUTPUT:
Deleted Michael Schumacher/111
1
Rubens Barrichello/99

Borrar objetos compuestos

```
// SESIÓN 1
ObjectSet result=db.query(new Predicate() {
public boolean match(Car car){
return car.getModel().equals("Ferrari"); }
});
Car found=(Car)result.next();
db.delete(found);
result=db.get(new Car(null));
listResult(result);
```

OUTPUT:
1
BMW[Rubens Barrichello/99]

```
// SESIÓN 2
// retrieveAllPilotsQBE
Pilot proto=new Pilot(null,0);
ObjectSet result=db.get(proto);
listResult(result);
```

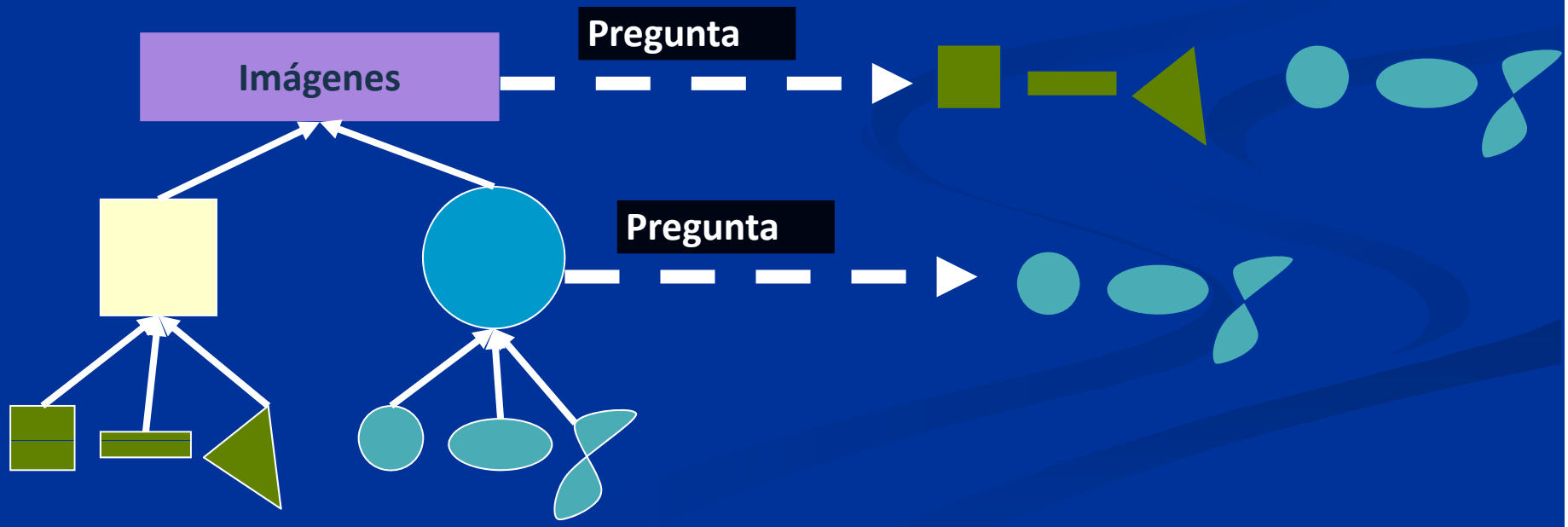
OUTPUT:
3
Somebody else/1
Rubens Barrichello/99
Michael Schumacher/100

No ha borrado el piloto "Somebody else", que era el piloto del Ferrari. Parece razonable.

```
// SI SE QUISIERA INDICAR QUE SE QUIEREN BORRAR EN CASCADA (MUCHO CUIDADO)
Db4o.configure().objectClass("Car").cascadeOnDelete(true);
```

Herencia

- Devuelve el tipo del objeto por el que se pregunta:
- Preguntando por la superclase, devuelve todos los objetos de las subclases
- Preguntando por la subclase, devuelve solamente los objetos de la subclase



Herencia

- ¿Qué sucede si al realizar la clase QBE es una clase abstracta o una interfaz?
  - No podemos utilizar un constructor para realizar el objeto "prototipo"
  - Solución: utilizar el nombre de la clase directamente

```
ObjectSet result=db.get(Pilot.class);
```

Transacciones simples

- db4o ofrece dos métodos
  - .commit() que termina la transacción indicando que se salven todos los cambios
  - .rollback() que indica que se deshaga la transacción
- Cuando se cierra la base de datos, se cierra implícitamente la transacción

Herramienta para comprobar los objetos en la base de datos db4o