

Java Data Objects

- Introducción
- Arquitectura
- Proceso de desarrollo
- Configuración
- Ejemplos
- Lenguajes de consulta
- Modelo de trabajo
- Ambientes

JDO - Java Data Objects

- Surje a partir del trabajo previo de ODMG (2001, versión 3.0).
- Es una definición, basada en interfaces, de persistencia de objetos para el lenguaje Java.
- Describe almacenamiento, consultas y recuperación de objetos de los repositorios.
- Actualmente se encuentra en la versión 3.0.
- No es una definición completa.

JDO - Java Data Objects

- Maneja de manera transparente el mapeo de las instancias al repositorio subyacente.
- Es transparente para las instancias que se están persistiendo.
- Se puede utilizar para persistir instancias contra varios paradigmas de almacenamiento (archivos, xml, bases de datos relacionales, bdoos).
- El estándar especifica que debe existir compatibilidad entre las diferentes implementaciones.

JDO - Java Data Objects

- Objetivos de JDO:
 - Cero o casi ninguna restricción al construir las clases.
 - No crear nuevos tipos.
 - No crear un nuevo lenguaje de acceso a la información.
 - No reemplazar a JDBC.
 - Que se pueda utilizar en varios entornos:
 - J2ME: dispositivos portátiles, etc., como aplicación embebida.
 - J2SE: cliente - servidor o aplicaciones desktop.
 - J2EE: enterprise java beans.

JDO - Java Data Objects

- Serialización:
 - Permite codificar los objetos de manera que se puedan enviar a un stream a disco o a través de la red.
 - Es estándar en cada ambiente en donde se ejecuta Java.
 - Fácil de usar.
 - Utiliza las clases de Java como modelo de datos.
 - No soporta persistencia realmente.
 - No presenta escalabilidad.

JDO - Java Data Objects

- JDBC:

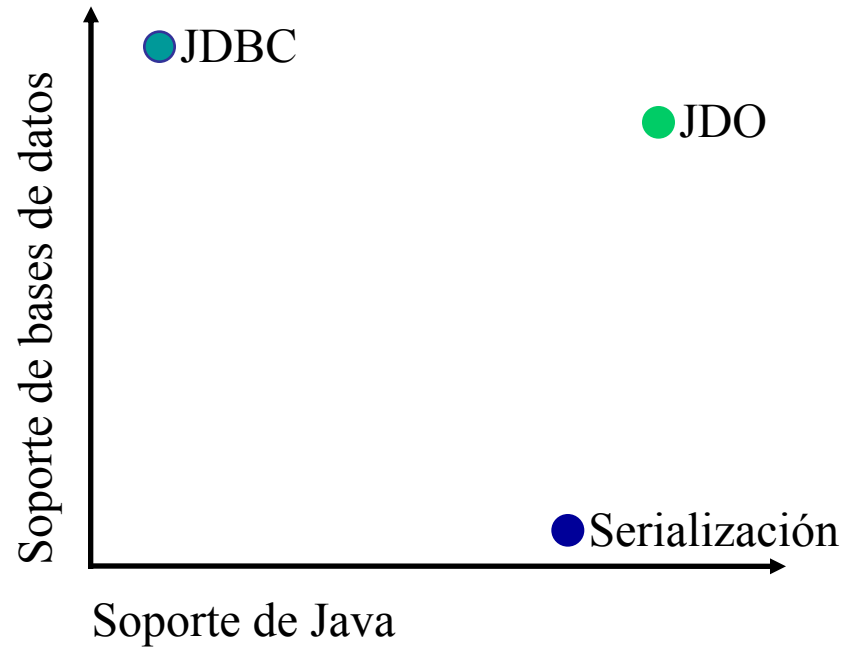
- Provee acceso a SQL.
- Utiliza un modelo de datos relacional.
- Las operaciones se expresan en términos de tablas, filas y columnas.
- Raramente portable entre productos.
- No soporta clases de Java.
- Tiende al paradigma procedural en vez del orientado a objetos.
- Procesamiento en lenguajes relacionados con SQL en vez de Java.

JDO - Java Data Objects

- JDO:
 - Soporta clases de Java.
 - Soporta transacciones.
 - Modelo de datos orientado a objetos.
 - Transparente para el programador.
 - Compatible entre diferentes productos.
 - Basado en meta data.
 - Mecanismo de consulta basado en Java.
 - Estándar muy reciente.

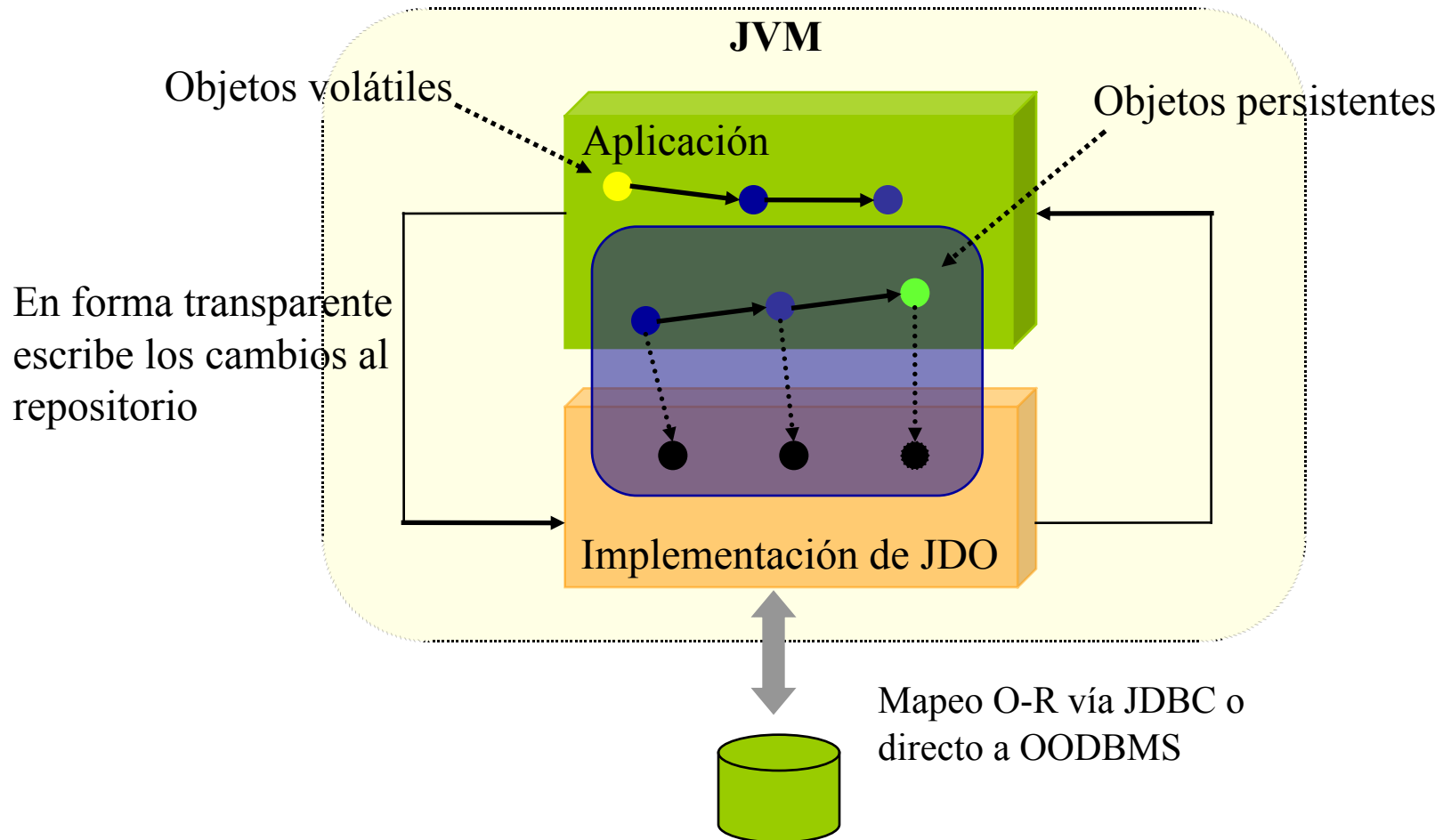
JDO - Java Data Objects

- Comparación entre las tres alternativas:



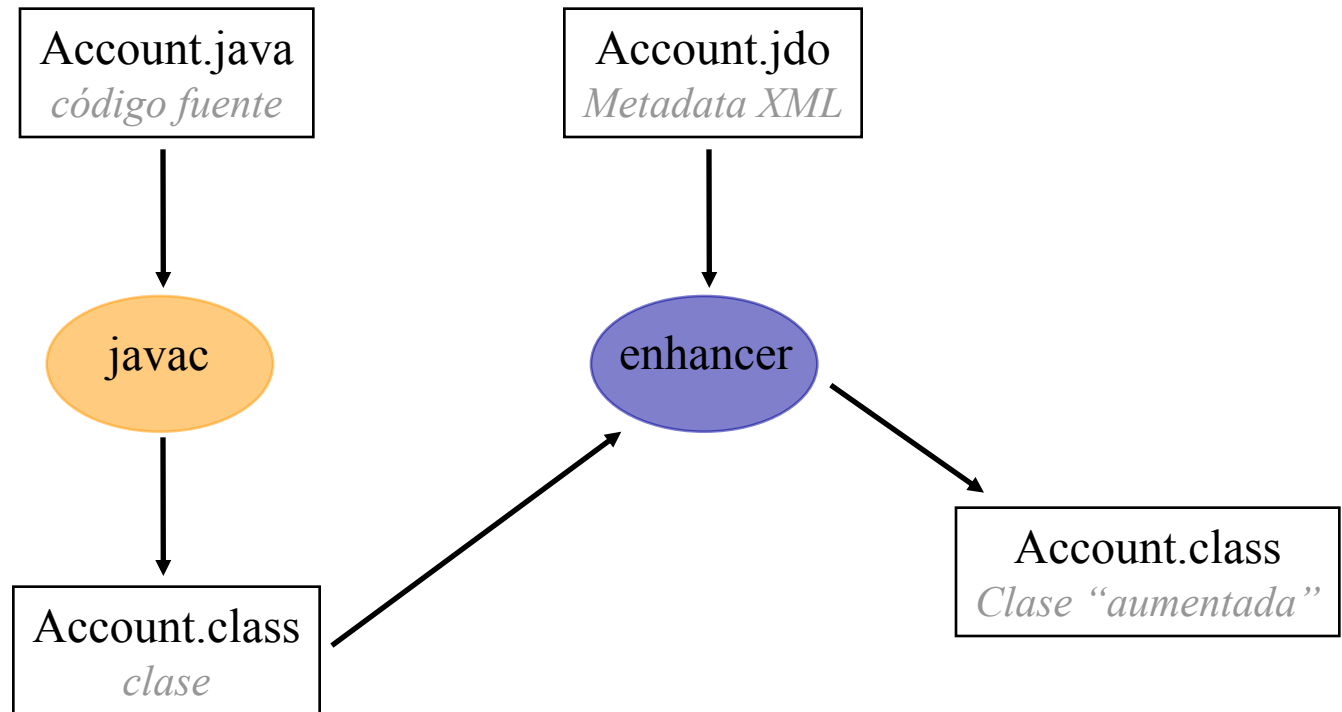
JDO - Java Data Objects

- Funcionamiento de JDO:



JDO - Java Data Objects

- Modelo de trabajo con JDO:



JDO - Java Data Objects

- JDO Enhancement:

Clase Java original

```
package org.apache.jdo.test;
public class A {
    long id;
    String name;
    B b;

    public A(String name) { this.name = name; }
    public void setId(long id) { this.id = id; }
    public void setB(B b) { this.b = b; }
    public String getName() { return name; }
    public B getB() { return b; }
    public long getId() { return id; }
    public String toString() { return "A : id=" + id + " [" + name + "] b=\\"" + b + "\""; }
}
```

43 líneas de código

Clase Java “aumentada”

486 líneas de código!!

JDO - Java Data Objects

- JDO Enhancer:

- Lee bytecodes y genera nuevos bytecodes para permitir a la implementación JDO realizar transparentemente:

- Recuperación de objetos.

- Controlar los cambios a las instancias persistentes.

- Escribir los cambios al repositorio.

- Esto permite que el programador no necesite explícitamente recuperar o guardar las instancias.

JDO - Java Data Objects

- Elementos importantes de JDO:
 - PersistenceManagerFactory (Interface):
 - Permite crear persistenceManagers.
 - PersistenceManager (Interface):
 - Es la puerta de entrada al repositorio.
 - Un persistenceManager encapsula una conexión al repositorio.

JDO - Java Data Objects

- Elementos importantes de JDO (cont.):
 - Transaction (Interface):
 - Demarca el espacio de trabajo transaccional con el repositorio.
 - Query (Interface):
 - Permite recuperar las instancias persistentes.

JDO - Java Data Objects

- Modelo de trabajo:
 - Usar una `PersistenceManagerFactory` para obtener un `PersistenceManager`.
 - Crear a partir del PM una transaction.
 - Usar la transaction para controlar los límites de la actividad contra el repositorio.
 - Trabajar con las instancias persistentes.
 - Cerrar la transaction.
 - Cerrar el `persistenceManager`.

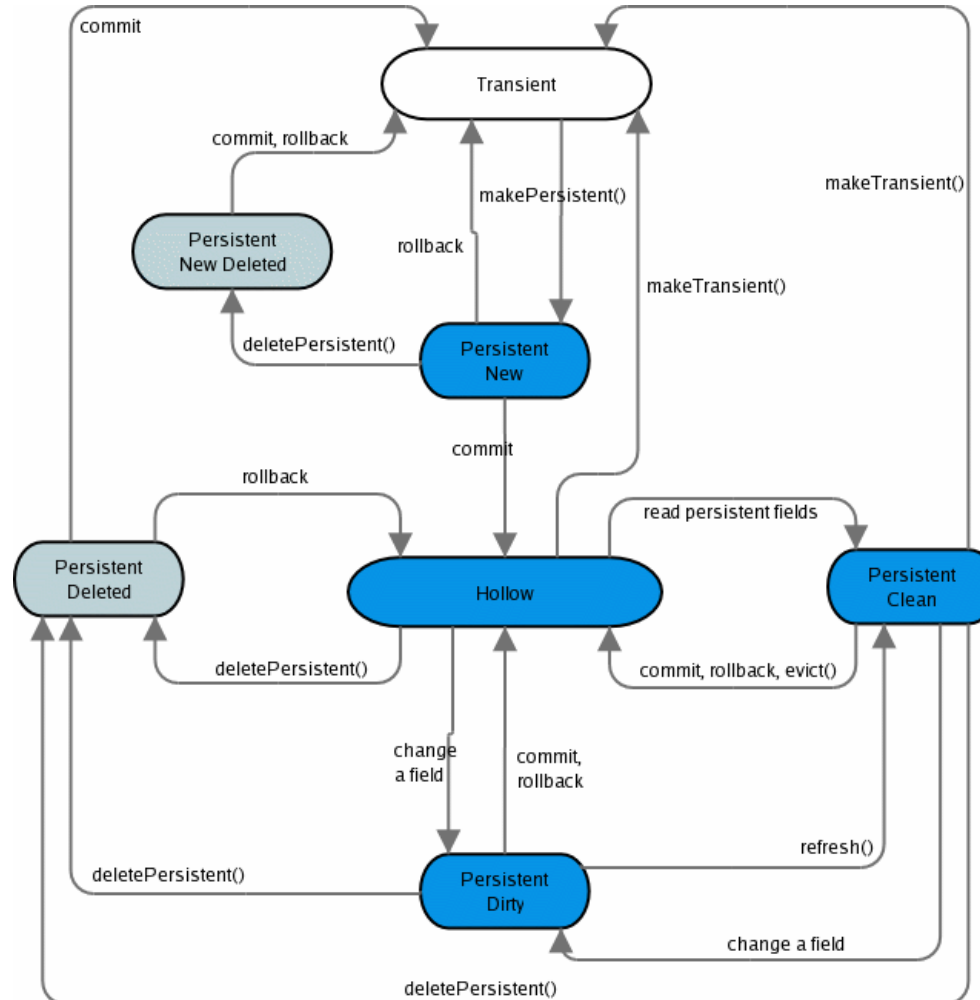
JDO - Java Data Objects

- A partir del modelo de trabajo, un objeto podría estar en alguno de los siguientes estados:

Name	Description
Transient	Any object created by the developer that do are not persisted. These don't have a JDO identity.
Persistent New	Any object that is newly persisted in the current transaction. A JDO identity has been assigned to these objects.
Persistent Dirty	Any persistent object that has been changed in the current transaction.
Hollow	Any persistent object that represents data in the datastore, but whose values are not in the instance.
Persistent Clean	Any persistent object that represents data in the datastore, and whose values have not been changed in the current transaction.
Persistent Deleted	Any persistent object that represents data in the datastore, and that has been deleted in the current transaction.
Persistent New Deleted	Any object that have been newly made persistent and then deleted in the same current transaction.
Persistent Non transactional	Any persistent object that represents data in the datastore, whose values are loaded but not transactionally consistent.
Persistent Non transactional Dirty	Any persistent object that represents data in the datastore, whose values are loaded but not transactionally consistent, and that has been modified.
Transient Clean	Any transient object that represents a transactional instance whose values have not been changed in the current transaction.
Transient Dirty	Any transient object that represents a transactional instance whose values have been changed in the current transaction.
Detached Clean	Any detached object that represents a persistent instance whose values have not been changed since detaching.
Detached Dirty	Any detached object that represents a persistent instance whose values have been changed since detaching.

JDO

- Estados de las instancias



JDO

- Tipos de clases
 - Persistence capable
 - Son clases cuyas instancias pueden ser almacenadas por JDO.
 - JDO realiza cambios en la clase compilada para permitir la persistencia de las instancias.

```
<class name="MyClass">  
    ...  
</class>
```

- Persistence aware
 - Son clases que no tienen instancias persistentes, pero que de todos modos son aumentadas por JDO.
 - Usualmente se utilizan para administrar las instancias de las clases persistentes.

```
<class name="MyClass" persistence-modifier="persistence-aware"/>
```

- Normal
 - Son clases que no son alteradas por el enhancer de JDO.

JDO - Java Data Objects

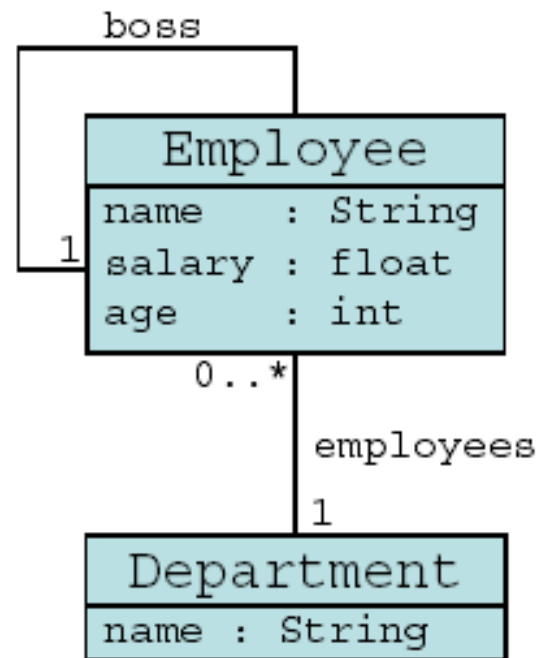
- Métodos de PersistenceManager:
 - Relacionados con la identidad de las instancias:
 - Object getId(Object instance)
 - Object getObjectByld (Object oid)
 - Relacionados con el manejo de las instancias:
 - void makePersistent (Object pc)
 - void deletePersistent (Object pc)
 - void makeTransactional (Object pc)
 - void makeNontransactional (Object pc)

JDO - Java data Objects

- Interface Transaction:
 - Se obtiene a través de un PersistenceManager:
 - `public Transaction currentTransaction();`
 - Existen dos modos de trabajo con las transacciones:
 - **Explícitamente** (`isActive()`, `begin()`, `commit()`..)
 - Implícitamente
 - Soportan diferentes modelos de concurrencia:
 - **Optimista**
 - Pesimista

JDO - Java Data Objects

- Un ejemplo simple:



JDO - Java Data Objects

- Un ejemplo simple:

```
public class Employee {  
    private String name;  
    private int age; private float salary; private Employee boss;  
    private Department department;  
  
    public Employee ( String name, int age ) {  
        this.name = name;  
        this.age = age;  
    }  
    public String getName ( ) {return name;}  
    ...  
    public Department getDepartment ( ) {return department;}  
    public void setDepartment ( Department d ) {department = d;}  
}
```

JDO - Java Data Objects

- Un ejemplo simple:

```
public class Department {  
    private String name; private Set employees = new HashSet();  
    public Department ( String name ) {  
        this.name = name;  
    }  
    public String getName ( ) {  
        return name;  
    }  
    ...  
    public void addEmployee ( Employee emp ) {  
        emp.setDeparment(this);  
        employees.add(emp);  
    }  
}
```

JDO - Java Data Objects

- Un ejemplo simple: Metadata XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
  <package name = "com.versant.jdoexample"/>
    <class name = "Employee"/>
      <class name = "Department">
        <field name = "employees">
          <collection element-type="Employee"/>
        </field>
      </class>
    </package>
  </jdo>
```


JDO - Java Data Objects

- Un ejemplo simple:

```
public static void main(String[] args) {  
    PersistenceManagerFactory pmf = PersistenceManagerFactories.getFactory();  
    ....  
    PersistenceManager pm = null;  
    try {  
        pm = pmf.getPersistenceManager();  
    } catch (Exception e) {  
        DatabaseAdministration.create(pmf.getConnectionURL(), null);  
        pm = pmf.getPersistenceManager();  
    }  
    Employee empleado= new Employee("pepe",45);  
    pm.currentTransaction.begin();  
    pm.makePersistent(empleado);  
    pm.currentTransaction.commit();  
    pm.close();}
```

JDO - Java Data Objects

- Cómo recuperamos a Pepe?
 - Existen 3 alternativas:
 - Mediante su oid.
 - Mediante el extent de la clase Employee.
 - Mediante un query.

JDO - Java Data Objects

- Recuperando a Pepe mediante su OID:

....

```
Employee empleado= new Employee("pepe",45);
```

```
pm.currentTransaction.begin();
```

```
pm.makePersistent(empleado);
```

```
//guardamos el oid de la nueva instancia persistente
```

```
session.setAttribute("id",pm.getObjectId(empleado));
```

...

```
pm.getObjectById(session.getAttribute("id"));
```

```
pm.currentTransaction.commit();
```

....

JDO - Java Data Objects

- Recuperando a Pepe mediante el extent:

```
Employee empleado= null;  
pm.currentTransaction.begin();  
Extent extent=pm.getExtent(Empleado.class);  
iterator=extent.iterator();  
while(iterator.hasNext()) {  
    empleado=(Empleado) iterator.next();  
    if (empleado.getName().equals("Pepe")){  
        //lo encontramos  
    }  
}  
pm.currentTransaction.commit();
```

JDO - Java Data Objects

- Recuperando a Pepe mediante un query:

```
Transaction tx = pm.currentTransaction();  
tx.begin();  
Extent employees = pm.getExtent(Employee.class);  
String filter = "name = aName";  
Query query = pm.newQuery(employees, filter);  
query.declareParameters("String aName");  
query.setOrdering("name ascending");  
Collection results = query.execute("Pepe");  
Employee pepe = (Employee) results.iterator().next();  
tx.commit();
```

Consultas con JDO

- JDO provee su propio lenguaje de consultas.
- Tiene como objetivo ser fácil para el programador Java.
- JDOQL tiene dos variantes
 - Forma declarativa
 - Se debe utilizar la clase Query y sus métodos
 - Se debe definir
 - Conjunto candidato a ser consultado
 - Filtros
 - Orden, etc
 - Forma basada en strings
 - Presente a partir de la versión 2 de JDO
 - Todo se especifica en un string que debe seguir cierto patrón

JDO - Java Data Objects

- Filtros en JDOQL:
 - Los filtros son expresiones booleanas.
 - Los identificadores son los atributos de las instancias.
 - La navegación se realiza a través del “.”.
 - Soporte para la navegación simple.
 - Soporte para la navegación de colecciones vía “contains()”.
 - Soporte de wildcards vía “startsWith()” y “endsWith()”.
 - Soporte para sustitución de parámetros y variables.

JDOQL

- Forma declarativa

```
Query q = pm.newQuery(org.jpox.Person.class, "lastName == \"Jones\" && age < age_limit");  
q.declareParameters("double age_limit");  
List results = (List)q.execute(20.0);
```

- Forma basada en String

```
Query q = pm.newQuery("SELECT FROM org.jpox.Person WHERE lastName == \"Jones\" +  
                        \" && age < :age_limit PARAMETERS double age_limit");  
List results = (List)q.execute(20.0);
```


JDOQL

- Forma declarativa

```
Query query = pm.newQuery("javax.jdo.query.JDOQL", "SELECT FROM org.jpox.MyClass WHERE param2 < threshold");
query.declareImports("import java.util.Date");
query.declareParameters("Date threshold");
query.setOrdering("param1 ascending");
List results = (List)query.execute(my_threshold);
```

- Algunos métodos importantes de la API

- `setClass()`: define el conjunto de instancias candidatas
- `setUnique()`: establece que el resultado debe ser único
- `setResult()`: define las proyecciones a realizar
- `setResultClass()`: define la clase del resultado
- `setFilter()`: establece el filtro a aplicar a las instancias
- `declareImports()`: importa las definiciones de las clases
- `declareParameters()`: declara todos los parámetros a utilizar
- `declareVariables()`: declara las variables que se utilizan

JDOQL

- Forma basada en String
 - Patrón de las consultas

```
SELECT [UNIQUE] [<result>] [INTO <result-class>]
      [FROM <candidate-class> [EXCLUDE SUBCLASSES]]
      [WHERE <filter>]
      [VARIABLES <variable declarations>]
      [PARAMETERS <parameter declarations>]
      [<import declarations>]
      [GROUP BY <grouping>]
      [ORDER BY <ordering>]
      [RANGE <start>, <end>]
```

- Ejemplo

```
SELECT UNIQUE FROM org.jpox.samples.Employee ORDER BY departmentNumber
```

JDOQL

- Utilización de parámetros

```
Query query = pm.newQuery(org.jpox.samples.store.Product.class, "price < limit");  
query.declareParameters("double limit");  
query.setOrdering("price ascending");  
List results = (List)query.execute(150.00);
```

```
Query query = pm.newQuery("SELECT FROM org.jpox.samples.store.Product WHERE " +  
    "price < limit PARAMETERS double limit ORDER BY price ASCENDING");  
List results = (List)query.execute(150.00);
```

JDOQL

- Consultas nombradas
 - Todos los ejemplos anteriores tienen el problema de estar “hardcodeados”.

```
Query query = pm.newNamedQuery(org.jpox.samples.company.Employee.class, "SalaryBelow12");  
List results = (List)query.execute();
```

```
<query name="SalaryBelow12" language="javax.jdo.query.JDOQL"><![CDATA[  
    SELECT FROM org.jpox.samples.company.Employee WHERE salary < 12 ORDER BY salary ASC  
]]></query>
```

JDOQL

- Especificación de resultados
 - En JDOQL es posible especificar diferentes resultados a partir de una consulta
 - this, equivale a retornar cada una de las instancias
 - el nombre un campo
 - Una variable
 - Un parámetro
 - Una función de agregación (avg, max, etc)
 - Una expresión de navegación (campo1.campo2)

JDOQL

- Tipos posibles para un resultado

- Object

- Se obtiene cuando el resultado contiene un solo elemento
 - Se utilizó UNIQUE o una función de agregación por ej. max(campo1).

- Object[]

- Se obtiene cuando el resultado es único, pero con más de una columna, por ej. max(campo1),avg(campo2)...

- List<Object>

- Se obtiene cuando existe una sola columna en el resultado y no se utilizan funciones de agregación, por ejemplo select campo1 ...

- List<Object[]>

- Se obtiene cuando existe más de una columna y no se utilizan funciones de agregación, por ejemplo select campo1, campo2 ...

JDOQL

- Subconsultas
 - Simples

```
SELECT FROM org.jpox.Employee WHERE salary >
    (SELECT avg(salary) FROM org.jpox.Employee e)
```

```
Query averageSalaryQuery = pm.newQuery(Employee.class);
averageSalaryQuery.setResult("avg(this.salary)");
```

```
Query q = pm.newQuery(Employee.class, "salary > averageSalary");
q.declareVariables("double averageSalary");
q.addSubquery(averageSalaryQuery, "double averageSalary", null, null);
List results = (List)q.execute();
```

- Referencia a la consulta exterior

```
SELECT FROM org.jpox.Employee WHERE salary >
    (SELECT avg(salary) FROM org.jpox.Employee e WHERE e.lastName == this.lastName)
```

JDO - Java Data Objects

- Ambientes JDO:
 - Existen básicamente dos maneras en las cuales JDO puede utilizarse:
 - “Non-managed Environments”:
 - Cliente/Servidor, “2” capas.
 - Manejo de las conexiones y transacciones en forma explícita.
 - “Managed Environments”:
 - Servidor de aplicaciones, Spring, n-capas.
 - Manejo de las conexiones de transacciones en forma implícita.

JDO - Java Data Objects






- Implementaciones conocidas:

Name	License	JDO Spec
DataNucleus Access Platform 	NonCommercial	1.0, 2.0, 2.1, 2.2, 2.3
JDOInstruments 	NonCommercial	1.0
JPOX 	NonCommercial	1.0, 2.0, 2.1
Kodo 	Commercial	1.0, 2.0
ObjectDB for Java/JDO 	Commercial	1.0, 2.0
Objectivity 	Commercial	1.0
Orient 	Commercial	1.0
hywy's PE:J 	Commercial	1.0
SignSoft intelliBO 	Commercial	1.0
Speedo 	NonCommercial	1.0
TJDO 	NonCommercial	1.0
Versant 	Commercial	1.0, 2.0
Xcalia 	Commercial	1.0, 2.0

JDO

- Implementaciones

- De referencia

- **JDO 1.0** : [FOStore](#) 
 - **JDO 2.0** : [JPOX 1.1](#) 
 - **JDO 2.1** : [JPOX 1.2](#) 
 - **JDO 2.2** : [DataNucleus AccessPlatform 1.0.1](#) 
 - **JDO 3.0** : [DataNucleus AccessPlatform 2.1.0](#) 

- Actuales

Name	License	JDO Spec
DataNucleus Access Platform 	NonCommercial	1.0, 2.0, 2.1, 2.2, 3.0
JDOInstruments 	NonCommercial	1.0
JPOX 	NonCommercial	1.0, 2.0, 2.1
Kodo 	Commercial	1.0, 2.0
ObjectDB for Java/JDO 	Commercial	1.0, 2.0
Objectivity 	Commercial	1.0
Orient 	Commercial	1.0
hywy's PE:J 	Commercial	1.0
SignSoft intelliBO 	Commercial	1.0
Speedo 	NonCommercial	1.0
TJDO 	NonCommercial	1.0
Versant 	Commercial	1.0, 2.0
Xcalia 	Commercial	1.0, 2.0

JDO

- Datanucleus

