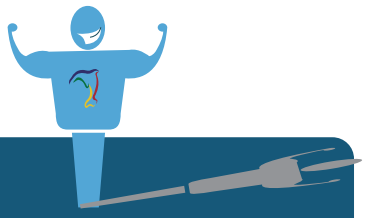
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

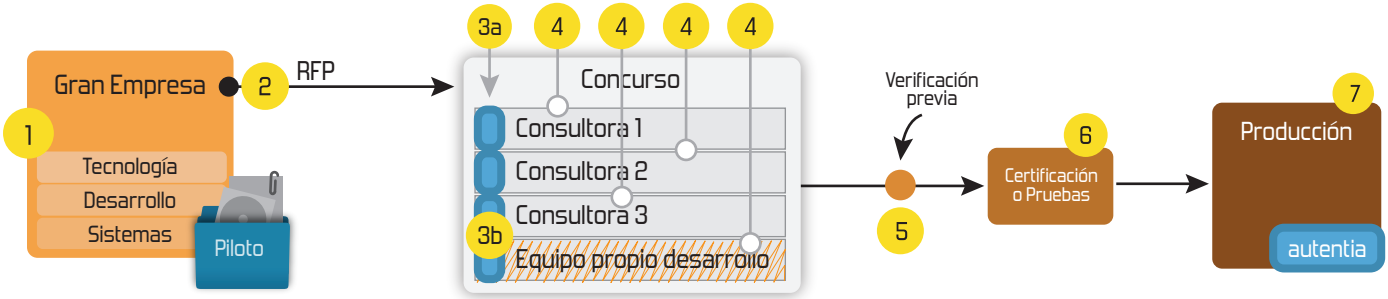
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por:
Roberto Canales Mora 2003-2005
Creador de AdictosAlTrabajo.com y

Director General de Autentia S.L.

Descargar este documento en formato PDF jdo.pdf

Introducción a JDO y OJB

Desde hace años, hemos vistos como distintas soluciones para mapear nuestras clases persistentes a bases de datos, no han acabado de consolidarse ....

La única diferencia actual, respecto a los modelos anteriores, es que parece que se ha realizado una normalización a nivel internacional.

Bajo el JCP o Java Community process JSR-12 se ha definido las especificación de como deben ser la implementaciones de JDO. Nosotros vamos a utilizar una implementación gratuita de apache llamada ObJectRelationalBridge (OJB).

Me ha costado un poco montarlo (muchas más horas de lo que me hubiera gustado) porque las instrucciones no son excesivamente intuitivas (jejejeje, ya lo veréis) pero hay que destacar que, el resultado final, me ha sorprendido gratamente..... No os asustéis porque aunque os describo los pasos, al final, podemos saltarnos casi todos .....

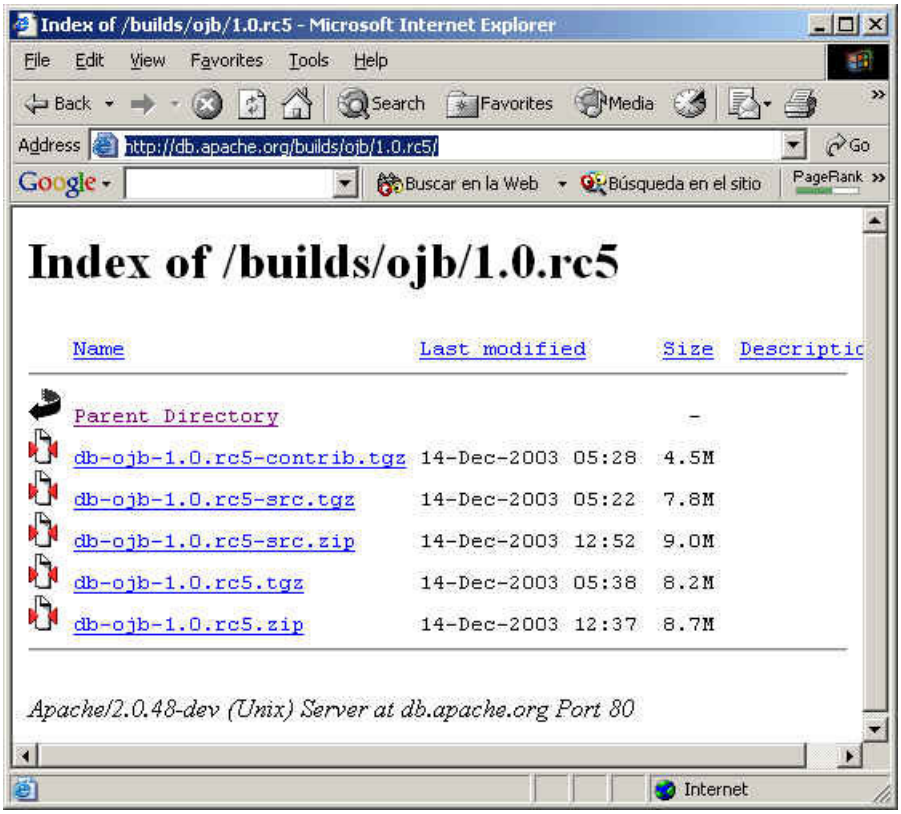
La gracia de este sistema es que, una clase normal, sin realizar transformaciones sobre ella, puede mapearse a una base de datos y permitir una manipulación trivial de datos.

Resumiendo lo que vamos a hacer:

- Instalamos el entorno
- Generamos el esqueleto de las futuras aplicaciones (ojb-blank.jar)
- Construimos nuestra clase persistente (Tutoriales.java)
- Construimos la clase que utiliza la persistente (Test.java)
- Modificamos los ficheros de configuración de OJB
- Empaquetamos el resultado
- Generamos las tablas en base de datos
- Desplegamos el resultado
- Ejecutamos la prueba

Descarga de OJB

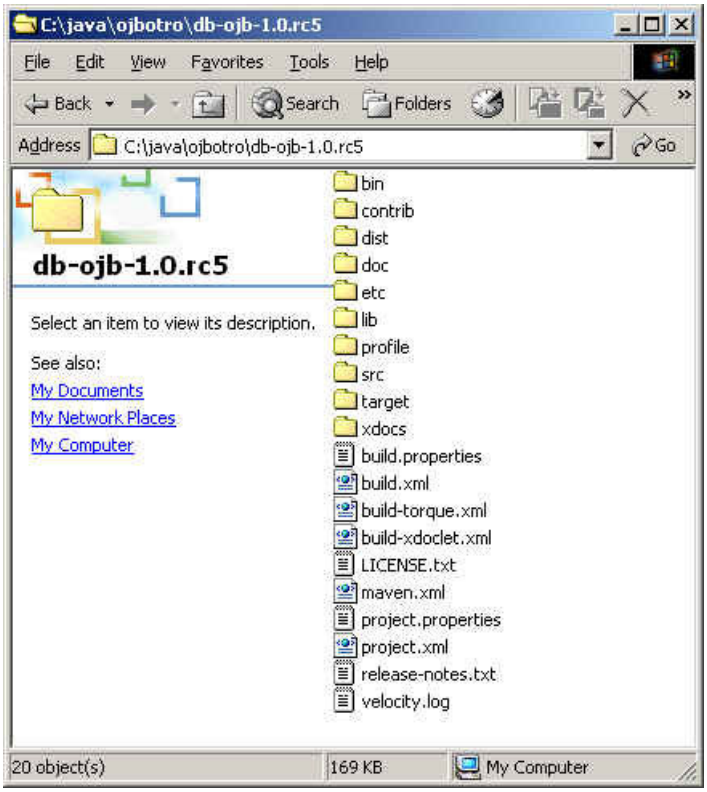
Como siempre, vamos a empezar descargándonos el entorno y haciendo un ejemplo simple.



Nos bajamos la versión binaria en ZIP y la descomprimimos.

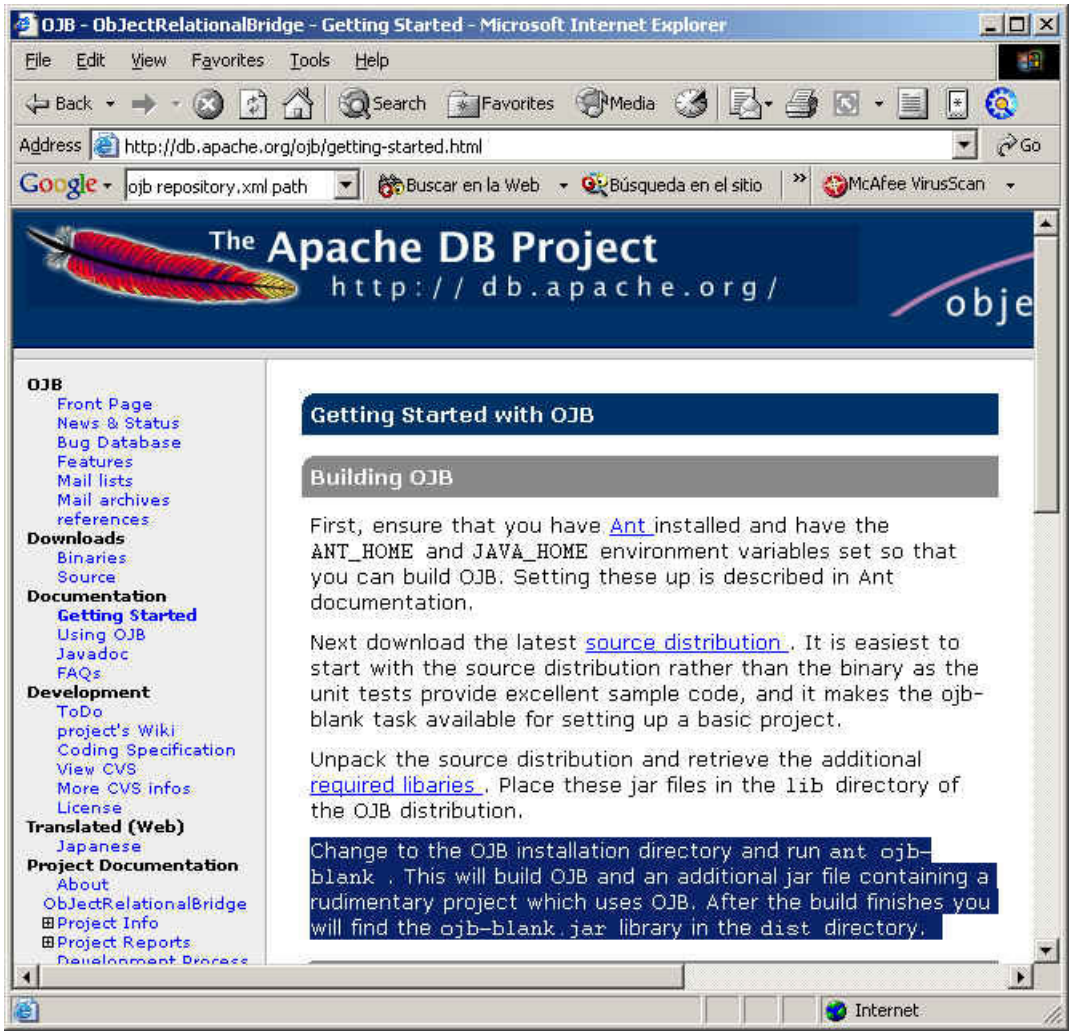
También descargaremos los fuentes, ya que queremos obtener de ellos un fichero con la estructura básica de un proyecto (ojb-blank.jar).

Éste es el aspecto una vez descomprimidos ambos zips (código y binarios) ...



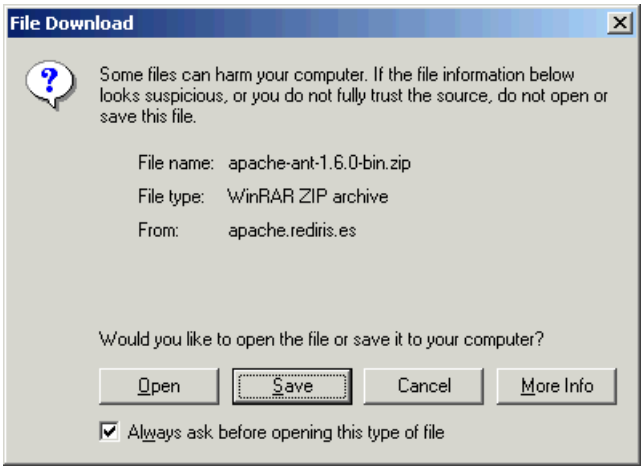
Vamos a seguir el tutorial y contaros como hemos ido arreglando los problemas que nos hemos encontrado http://db.apache.org/ojb/getting-started.html.

En tutorial nos guía pero requiere mucho esfuerzo e imaginación.



Descarga de ANT

Es necesaria una versión de ANT (nos hemos descargado la versión 1.6). Podéis recordar como se hacia en otro de nuestros tutoriales.



Recordar que hay que establecer la variable de entorno ANT_HOME e incluir el trayecto de ant en el path.

Generación de ojb-blank.jar

Seguimos las instrucciones para generar el paquete de ojb-blank.jar

Para ello, ejecutamos el comando:

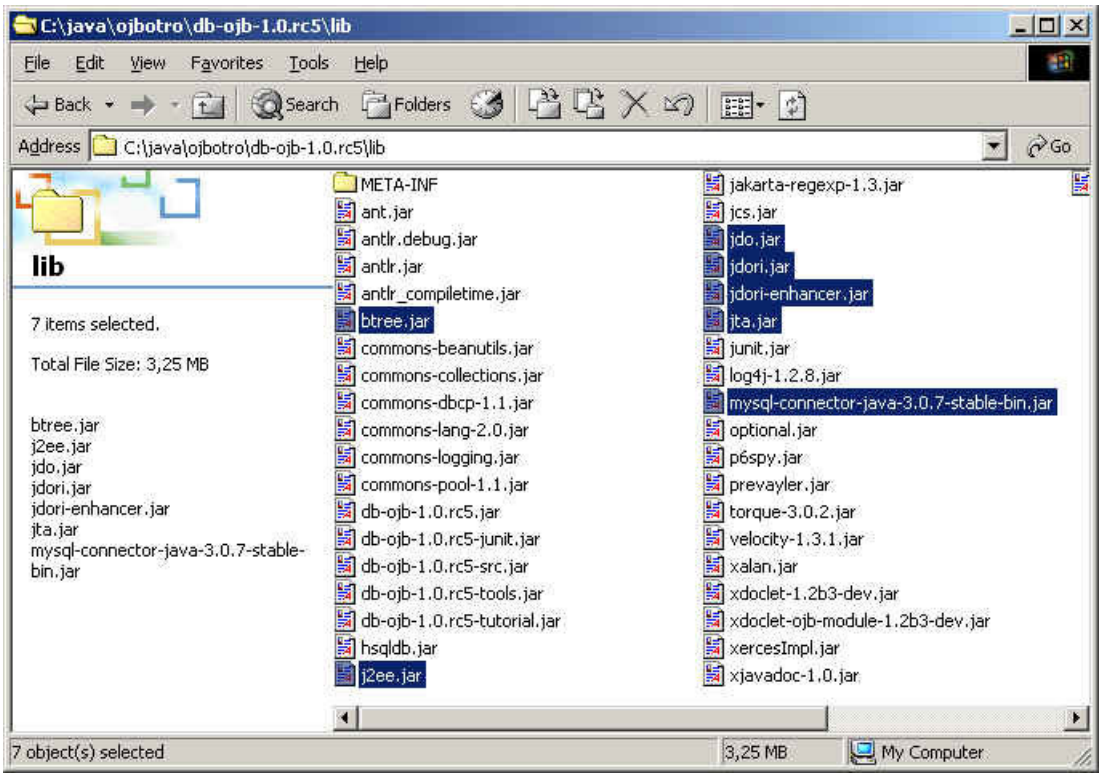
```
ant ojb-blank
```

Y nos falla... porque nos faltan ficheros jar ... (ejecutar **ant ojb-blank -verbose** para más información)

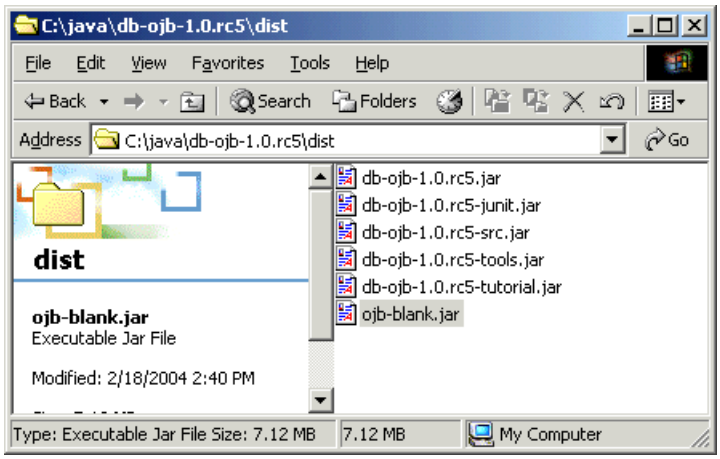
Debemos ir al siguiente enlace para ver donde encontrarlos:

http://db.apache.org/ojb/dependencies.html

Para que nos funcione tenemos que obtener los siguiente ficheros (seguir los enlaces de j2ee, jta y jdo)

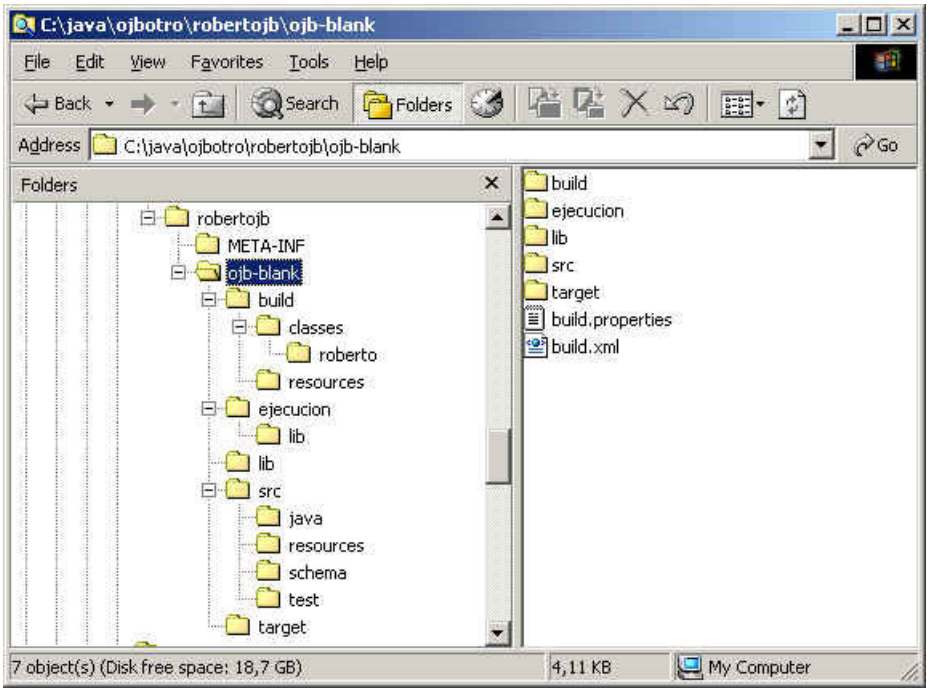


Si volvemos ahora al comando ant, ya nos aparece el fichero ojb-blank.jar.



Vamos a descomprimir el fichero en un directorio.

En el futuro ya no tendremos que repetir lo anterior.



Código de nuestra aplicación

Y vamos a crear la clase que queremos que represente una tabla en base de datos (la copiaremos en el directorio src).

Es una clase completamente normal (Diríamos que es como un VO, Value Object).

```
package roberto;

/**
 * Tutoriales.java
 *
 * Created on February 19, 2004, 1:16 PM
 * @author  Roberto Canales
 */
public class Tutoriales {

    private int id;
    private String titulo;
    private String descripcion;
    private String enlace;

   public Tutoriales() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public java.lang.String getTitulo() {
        return titulo;
    }

    public void setTitulo(java.lang.String titulo) {
        this.titulo = titulo;
    }

    public java.lang.String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(java.lang.String descripcion) {
        this.descripcion = descripcion;
    }

    public java.lang.String getEnlace() {
        return enlace;
    }

    public void setEnlace(java.lang.String enlace) {
        this.enlace = enlace;
    }

}
```

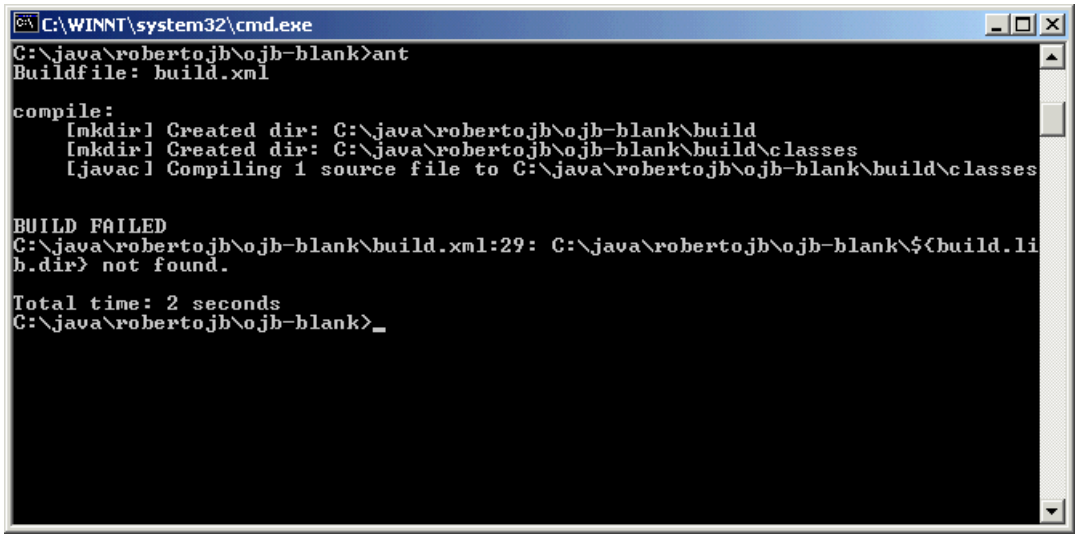
Ahora creamos la clase de prueba que utiliza el API, para acceder de un modo transparente a las capacidades persistentes.

```
/*
 * Test.java
 * Created on February 19, 2004, 2:03 PM
 */

package roberto;

import java.util.*;
import javax.jdo.*;
import org.apache.ojb.broker.*;
import org.apache.ojb.broker.query.*;
/**
 *
 * @author  Roberto Canales
 */
public class Test {

   public static void main(String[] args) {

      PersistenceBroker broker = PersistenceBrokerFactory.defaultPersistenceBroker();

      System.out.println("Lista de los tutoriales");

      org.apache.ojb.broker.query.Query query = new QueryByCriteria(Tutoriales.class, null);
      try
      {
         Collection allProducts = broker.getCollectionByQuery(query);

         java.util.Iterator iter = allProducts.iterator();
         while (iter.hasNext())
         {
            Tutoriales aux = (Tutoriales)iter.next();
            System.out.println("El id es " + aux.getId());
            System.out.println("El titulo es " + aux.getTitulo());
            System.out.println("La descripcion es " + aux.getDescripcion());
         }
      }
      catch (Throwable t)
      {
         t.printStackTrace();
      }
   }
}
```

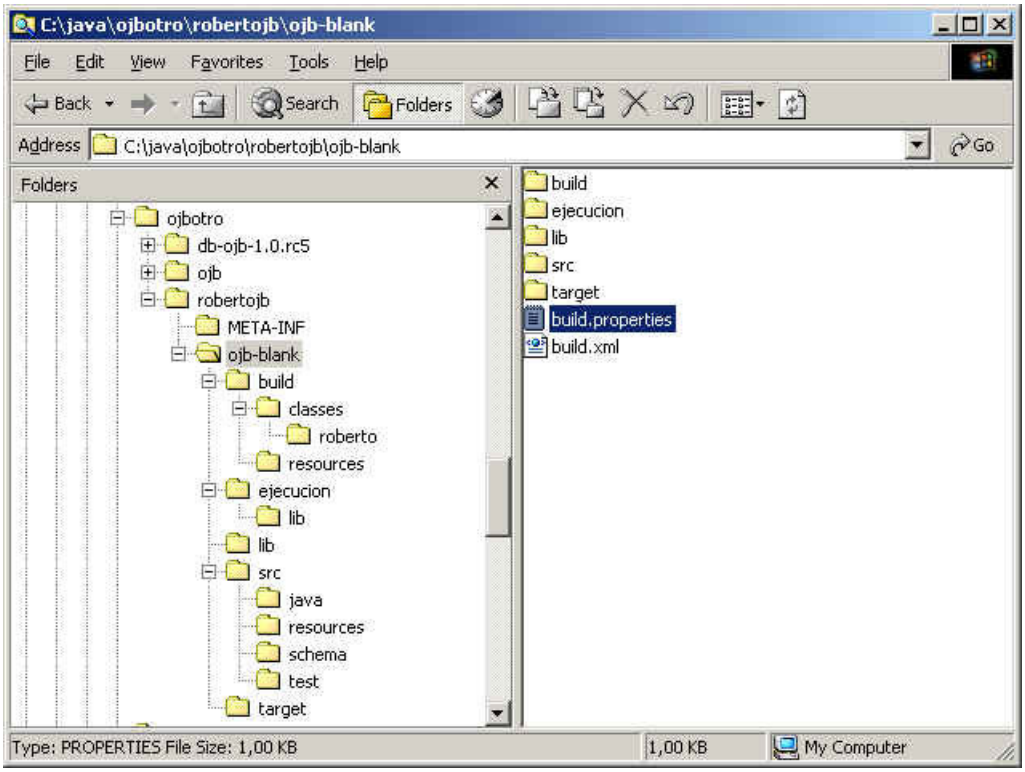
Compilación de la aplicación

Vamos a compilar el proyecto sobre el esqueleto creado por ojb-blank.

Si vamos al directorio que hemos elegido .... y ejecutamos en comando **ant** ..... falla ...



Hay que hacer un pequeño cambio en los ficheros para que compile .... aprovechamos y hacemos alguno más que nos sea interesante.



El **build.properties** posee variables sacadas del **build.xml**.... (las ajustamos a MySql)

```
###############################################################################
# These are build properties for the ojb-blank project. In addition to
# any customization of your build process, you will probably want
# to change the values in the block below to map to your database
# rather than a generic HSQLDB instance.
#
jcdAlias=default
dbmsName=mysql
jdbcLevel=2.0
jdbcRuntimeDriver=com.mysql.jdbc.Driver
urlProtocol=jdbc:mysql://localhost/tutoriales
urlSubprotocol=
urlDbalias=tutoriales
databaseUser=root
databasePassword=
databaseHost=172.0.0.1

# Set this to the name of the jar file you want produced from the jar target
jar.name=tutoriales.jar

###############################################################################
# Build Properties
#
source.dir=src
source.java.dir=${source.dir}/java
source.resource.dir=${source.dir}/resources
source.test.dir=${source.dir}/test

build.dir=build
build.classes.dir=${build.dir}/classes/
build.lib.dir=${build.dir}/lib/
build.resource.dir=${build.dir}/resources/

lib.dir=lib

target.dir=target
```

También cambiamos el **build.xml** (el script de ant).

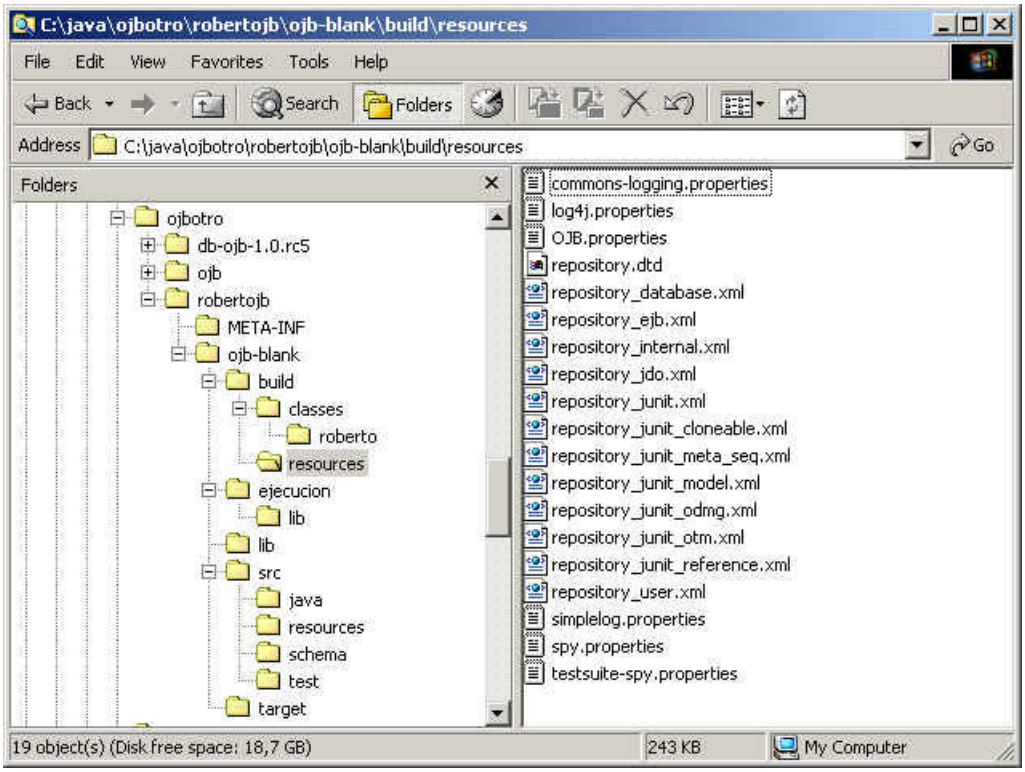
Al final hemos añadido un target para ayudar a la ejecución futura (incluir todos los jar en el classpath)

```
<project name="ojb-blank" default="build" basedir=".">

  <!--
     This build file is designed so that you can add build/classes, build/resources
     and lib/*.jar to a classpath and run the project.

     The default "build" target will build everything and copy src/resources/*
     over to build/resources. Class files will wind up in build/classes

     The Jar target kindly jars things up for you.

     You will probably want to modify this for your environment, but hopefully
```

```
    it will work well for you.

    Also, you will want to customize build.properties
  -->
  <property file="build.properties"/>

  <path id="compile-classpath">
    <fileset dir="${lib.dir}">
      <include name="**/*.jar"/>
    </fileset>
  </path>

 <path id="ejecucion-classpath">
     <fileset dir=".">
       <include name="**/*.jar"/>
       <include name="**/*.zip"/>
    </fileset>
    <pathelement path="build/resources"/>
    <pathelement path="."/>
  </path>


  <target name="compile">
    <tstamp/>
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${build.classes.dir}"/>
    <javac srcdir="${source.java.dir}" destdir="${build.classes.dir}">
      <classpath refid="compile-classpath"/>
    </javac>
  </target>

  <target name="build" depends="compile">
    <copy todir="${build.resource.dir}/">
      <fileset dir="${source.resource.dir}">
        <include name="*.properties"/>
        <include name="*.dtd"/>
        <include name="repository_*.xml"/>
        <exclude name="build.properties"/>
      </fileset>
      <filterset>
        <filter token="JCD_ALIAS" value="${jcdAlias}"/>
        <filter token="DBMS_NAME" value="${dbmsName}"/>
        <filter token="JDBC_LEVEL" value="${jdbcLevel}"/>
        <filter token="DRIVER_NAME" value="${jdbcRuntimeDriver}"/>
        <filter token="URL_PROTOCOL" value="${urlProtocol}"/>
        <filter token="URL_SUBPROTOCOL" value="${urlSubprotocol}"/>
        <filter token="URL_DBALIAS" value="${urlDbalias}"/>
        <filter token="USER_NAME" value="${databaseUser}"/>
        <filter token="USER_PASSWD" value="${databasePassword}"/>
      </filterset>
    </copy>
  </target>

  <target name="clean">
    <delete dir="target" quiet="true"/>
    <delete dir="build" quiet="true"/>
    <delete file="velocity.log"/>
  </target>

  <target name="jar" depends="build">
    <mkdir dir="${target.dir}"/>
    <jar index="true" jarfile="${target.dir}/${jar.name}">
      <fileset dir="${build.classes.dir}/">
        <include name="**/*.class"/>
      </fileset>
      <fileset dir="${build.resource.dir}">
        <include name="**/*"/>
      </fileset>
    </jar>
  </target>

  <target name="ejecuta" depends="">
    <java fork="no"
        classname="roberto.Test"
        failonerror="true"
        classpathref="ejecucion-classpath">
     </java>
  </target>

</project>
```

Hay que modificar otros ficheros para configurar el comportamiento del sistema de mapping.

Modificamos el fichero **repository_database.xml**




```
<!-- @version $Id: repository_database.xml,v 1.17 2003/11/27 16:42:12 arminw Exp $ -->
<!--
Define here all used connections.
One defined connection should be defined as the default one,
by set default-connection="true" - this could be done at runtime too.

It is possible to set user/password at
runtime or let login different users at runtime using the same
database. Use different PBKey with same jcdAlias name but
different user/password.

Ditto it is possible to add jdbc-connection-descriptor at runtime
using the MetadataManager.
-->

    <!-- this connection was used as the default one within OJB -->
    <jdbc-connection-descriptor
                            jcd-alias="default"
                            default-connection="true"
                            platform="mysql"
                            jdbc-level="2.0"
                            driver="com.mysql.jdbc.Driver"
                            protocol="jdbc:mysql://localhost/tutoriales"
                            subprotocol=""
                            dbalias="tutoriales"
                            username="root"
                            password=""
        eager-release="false"
                            batch-mode="false"
        useAutoCommit="1"
        ignoreAutoCommitExceptions="false">

        <connection-pool
            maxActive="21"
            validationQuery="" />

        <sequence-manager className="org.apache.ojb.broker.util.sequence.SequenceManagerHighLowImpl">
            <attribute attribute-name="grabSize" attribute-value="20"/>
            <attribute attribute-name="autoNaming" attribute-value="true"/>
            <attribute attribute-name="globalSequenceId" attribute-value="false"/>
            <attribute attribute-name="globalSequenceStart" attribute-value="10000"/>
        </sequence-manager>
    </jdbc-connection-descriptor>
```

Y tenemos que especificar como queremos que se resuelva el mapeo entre los elementos de la clase y las tablas, en el fichero **repository_user.xml**

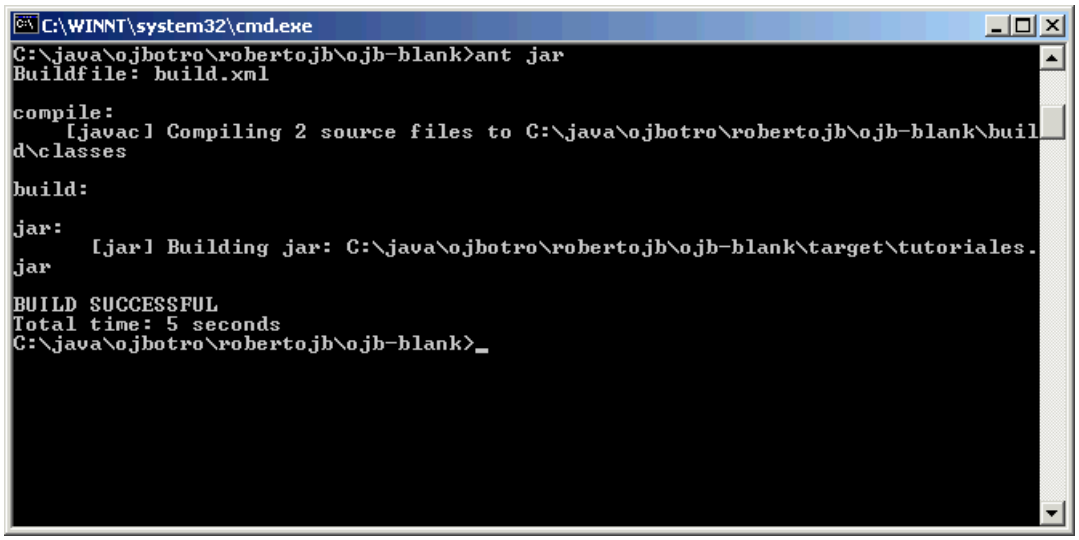
```
 <!-- Please keep user defined mappings in this file only
to avoid mixing user defined and system mappings. -->
<!-- Mapping of User defined classes starts here -->

<!-- The mappings for the tutorial classes are placed here to make it
easier to find them for OJB newbies.
Please remove them if you don't need them in your environment. -->
```

```
<!-- Definitions for org.apache.ojb.tutorial1.Product -->
<class-descriptor class="roberto.Tutoriales" table="TUTORIALESJDO">
<field-descriptor name="id" column="ID" jdbc-type="INTEGER" primarykey="true"
autoincrement="true" />
<field-descriptor name="titulo" column="TITULO" jdbc-type="VARCHAR" />
<field-descriptor name="descripcion" column="DESCRIPCION" jdbc-type="VARCHAR" />
<field-descriptor name="enlace" column="ENLACE" jdbc-type="VARCHAR"/>
</class-descriptor>
```

Y si compilamos y empaquetamos..... por fin funciona. Las clases compilan ....

El comando utilizado ahora es:

ant jar

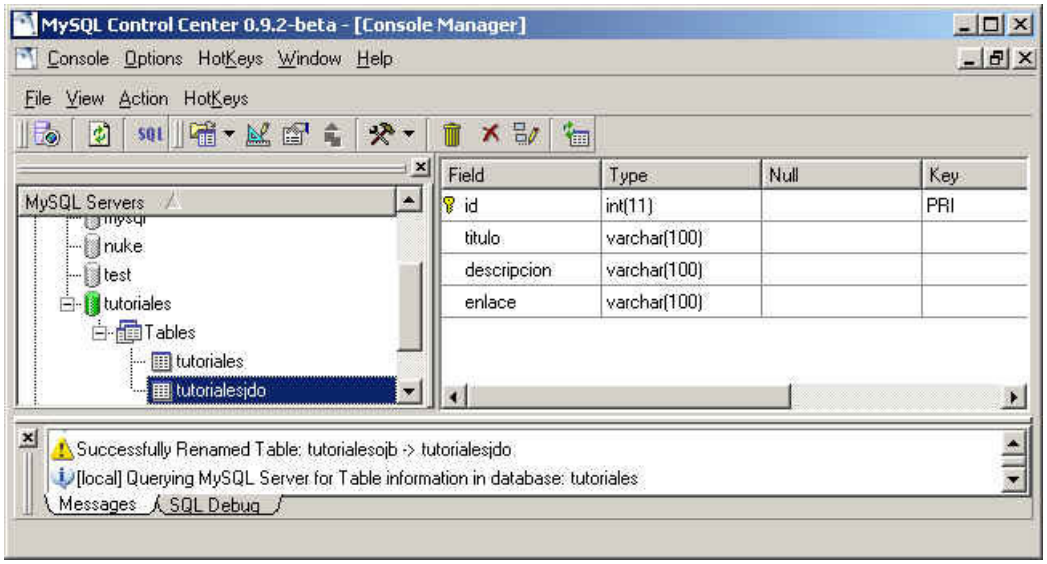


El fichero jar debería tener todo lo necesario para que la aplicación funcione (en ejecución) ..... aunque si lo intentamos, le falta el fichero **repository.xml**.... graciosamente, el más importante ....

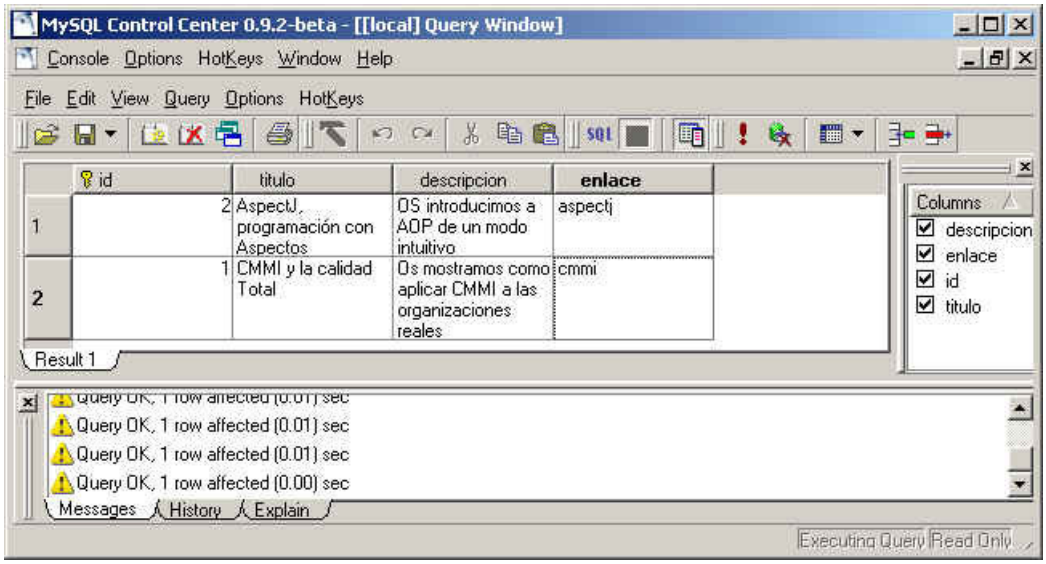
Lo localizaremos en nuestro ordenador y lo introduciremos en nuestro directorio de trabajo (para que en sucesivas ocasiones se empaquete con los demás).

Creación de la Base de Datos

Obviamente, necesitamos las tablas (las creamos a mano)

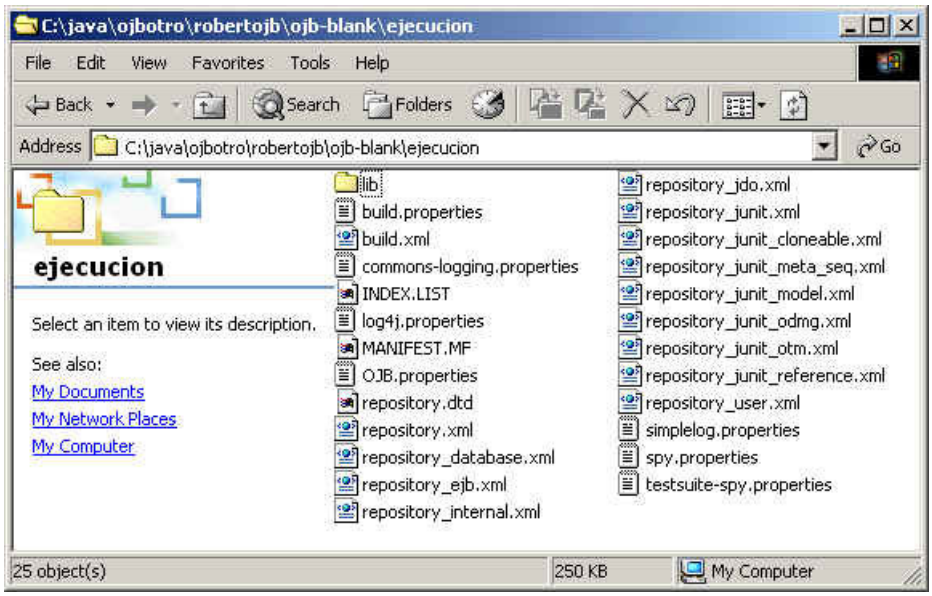


Insertamos algún dato.

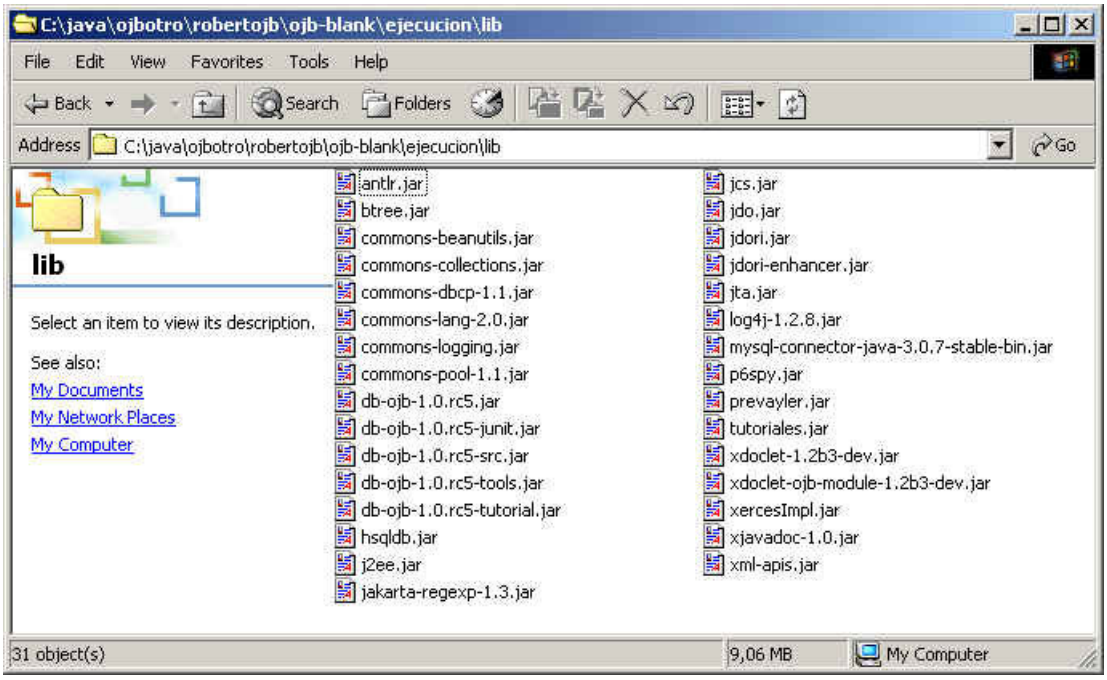


Ejecución

Vamos a descomprimir el fichero **jar** que hemos obtenido como el resultado.

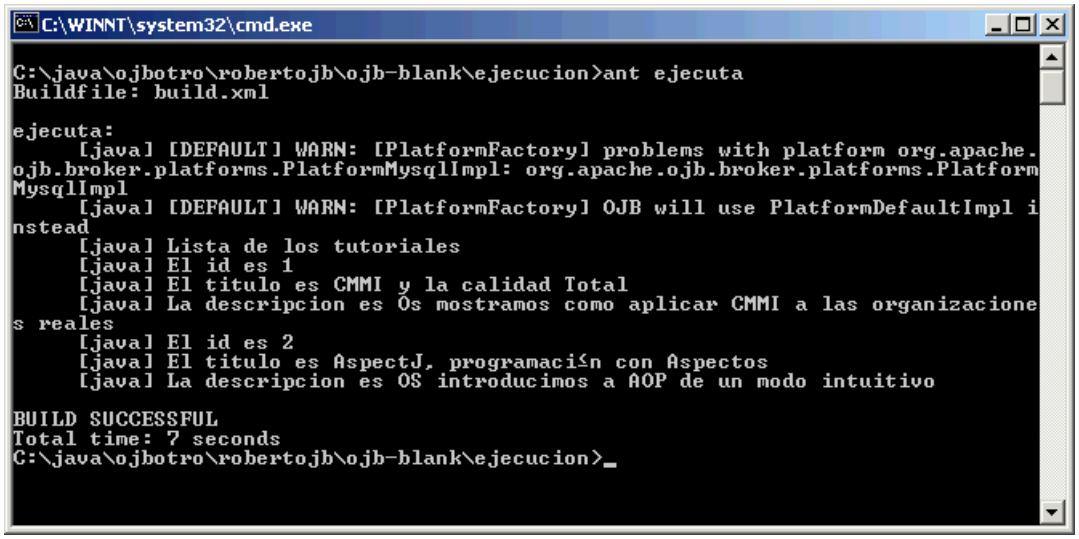


y también copiamos todas las librerías a un directorio **lib**.



Y podemos probar el programa, ejecutando (que es el target que añadimos anteriormente)

ant ejecuta



Conclusión

El montaje ha sido costoso pero si os quedais con el concepto.... solo hemos escrito dos clases relativamente sencillas y tenemos nuestro sistema JDO/OJB.

No os puedo comentar sobre el rendimiento pero ..... si no se ha transformado la clase original (como en otras soluciones JDO) nos da pistas que todo se realiza a través de reflexión en tiempo real.....

Ya os contaremos .....

Sobre el Autor ..

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Reingeniería JDO con Druid	Os mostramos como crear vuestras clases y descriptores JDO, de tablas existentes, con la herramienta gratuita Druid.
Creación automática de recursos Hibernate con Middlegen	En este tutorial aprendereis como utilizar la herramienta middlegen para generar distintas capas de persistencia (CMP 2.0, JDO, Hibernate, Torque), a partir de un modelo físico de datos, de un modo automático, mediante el uso de la herramienta middlegen
Generación automática de código JDBC	En este tutorial os enseñamos como, sin conocimiento de JDBC, crear vuestro programas en Java, gracias a JDBCTest.
CMP Entity Beans y MySql	Os mostramos como crear un Entity Bean con persistencia controlada por el servidor, configurado para usar MySql
JDBC y MySql	En el tutorial anterior vimos como instalar MySQL en Windows, ahora vamos a ver como acceder desde una aplicación Java.
Desarrollo de Entity Beans	Os mostramos como construir un Entity Bean básico y desplegarlo en el servidor J2EE de referencia. Lo usaremos como base de buenas prácticas J2EE
Web Services en tu IPAQ	Cesar Crespo nos enseña como programar accesos Web Services desde tu IPAQ en Visual C++ con PocketSOAP, Apache SOAP y Axis
AspectJ, Programación con Aspectos	Os mostramos como configurar AspectJ (extensión Java para la programación basada en aspectos) y un pequeño ejemplo para medir la velocidad de una función sin alterar su código.
Soporte de Asserts en Java 1.4.x	Os mostramos como utilizar los asserts en Java (disponibles a partir de la versión 1.4)
Novedades en Java 1.5	Ya está disponible la versión Beta del J2SDK 1.5. Os mostramos algunas de las nuevas características introducidas en el lenguaje Java: Clases genéricas, enumeraciones, bucles simplificados, etc.