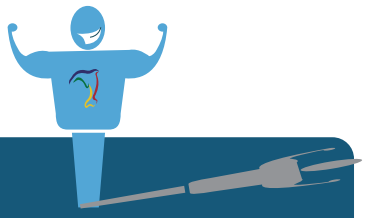
autentia

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...

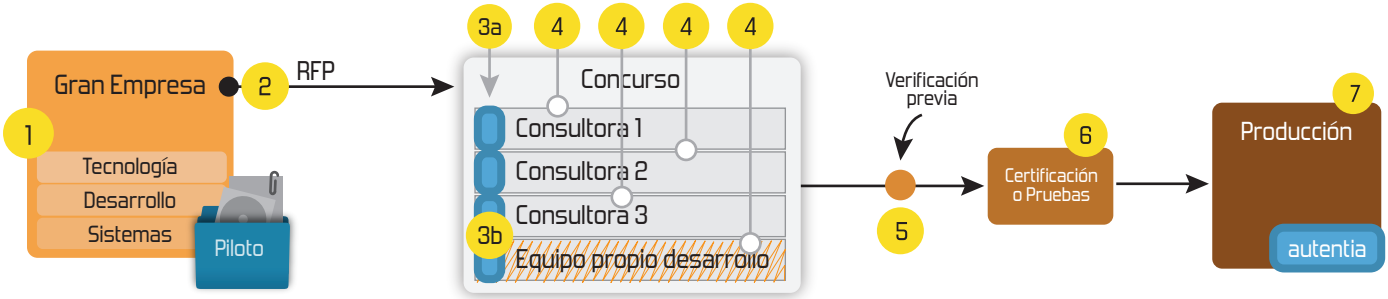
1. Desarrollo de componentes y proyectos a medida



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

adictos altrabajo Powered by autentia
Hosting Patrocinado por enREDados.com enREDados

Lanzado TNTConcept versión 0.6 (12/07/2007)

Desde Autentia ponemos a vuestra disposición el software que hemos construido (100% gratuito y sin restricciones funcionales) para nuestra gestión interna, llamado TNTConcept (au**T**e**NT**ia).

Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Acegi, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño ..... Saber más en: http://tntconcept.sourceforge.net/

Contacte con Cristóbal González
criskerberos-tutoriales@yahoo.com

Autor: Cristóbal González Almirón es consultor de desarrollo de proyectos informáticos.

Su experiencia profesional se ha desarrollado en empresas como Compaq, HP, Mapfre, Endesa, Repsol, Universidad Autónoma de Madrid, en las áreas de Desarrollo de Software (Orientado a Objetos), tecnologías de Internet, Técnica de Sistemas de alta disponibilidad y formación a usuarios.

NUEVO CATÁLOGO DE SERVICIOS DE AUTENTIA (PDF 6,2MB)

www.adictosaltrabajo.com es el Web de difusión de conocimiento de www.autentia.com

autentia real business solutions

Catálogo de cursos

Descargar este documento en formato PDF coregest2.pdf

Firma en nuestro libro de Visitas **<-----> Asociarme al grupo AdictosAlTrabajo en eConozco**

Master Experto Java
100% alumnos se colocan. Incluye Struts, Hibernate, Ajax
www.grupoatrium.com

Centro Oficial Sun JAVA
Master , Prep. Exa Cert. , Cursos Java SE, Java EE, J2ME, JSF AJAX
www.programia.es

Roundtrip Java 5
Apollo for Eclipse Modeling tool with integration of UML 2.1
www.Gentleware.com

Anuncios Google

Fecha de creación del tutorial: 2007-07-31

Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: Hibernate

Introducción

Este tutorial sigue la serie de tutoriales sobre MyFaces JSF y Maven, en entorno Eclipse. En esta ocasión vamos a introducir Hibernate 3.2 con anotaciones en nuestro proyecto, todo ello respetando la filosofía de Maven, e integrándolo en Eclipse. Algunas de las cosas que vamos a ver son:

- Añadir los módulos de hibernate a nuestros proyectos Maven
- Configurar Eclipse para utilizar Hibernate en proyectos Maven
- Configurar Hibernate en nuestro módulo de lógica de negocio
- Crear pruebas unitarias de persistencia para nuestro módulo de negocio,
- Crear una instancia de Tomcat para probar nuestra aplicación

Requisitos

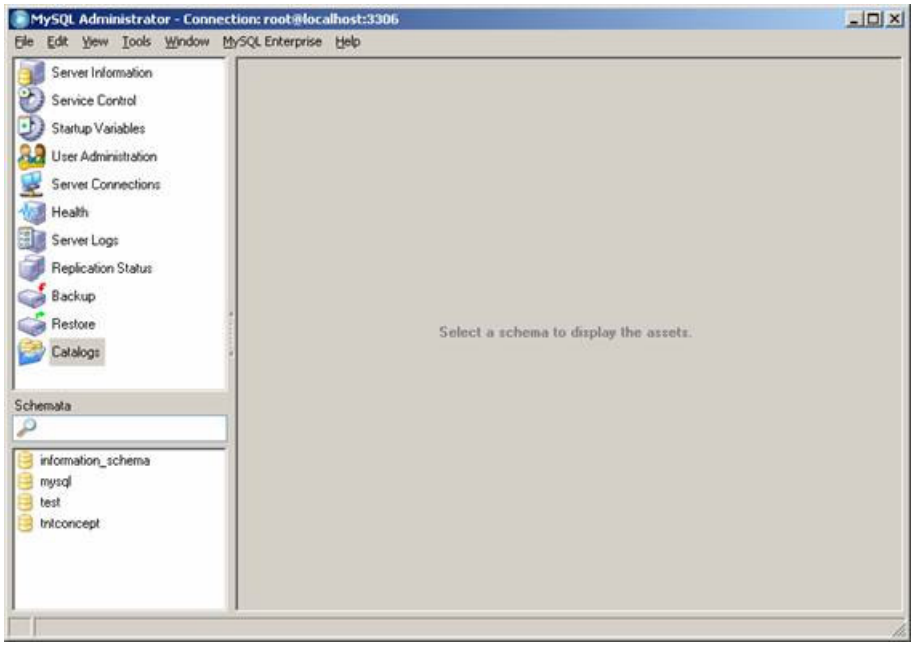
- Maven 2.0
- Eclipse 3.2 con el módulo WST y dentro de él, la opción de JSF. Se puede obtener con Callisto.
- Una base de datos SQL compatible con Hibernate. En nuestro ejemplo vamos a usar MySQL 5
- Hibernate tools para eclipse 3.2 instaladas. Ver el tutorial de Alex.
- Tutoriales publicados en AdictosAlTrabajo:
  - Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: aplicación multimódulo Tutorial sobre JSF y Maven
  - Proyecto con JSF Myfaces, Maven y
  - Hibernate Tools y la generación de código
  - Crear el sitio web de documentación del proyecto con Maven Site
  - Gestióin documental

Introducción de Hibernate en la aplicación

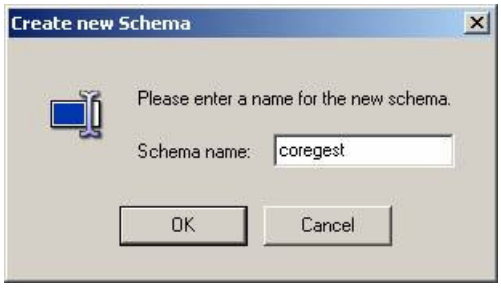
Creación de la base de datos de la aplicación

Lo primero que vamos a hacer es crear la base de datos coregest en la instancia de MySql local. Abrimos el administrador de MySql

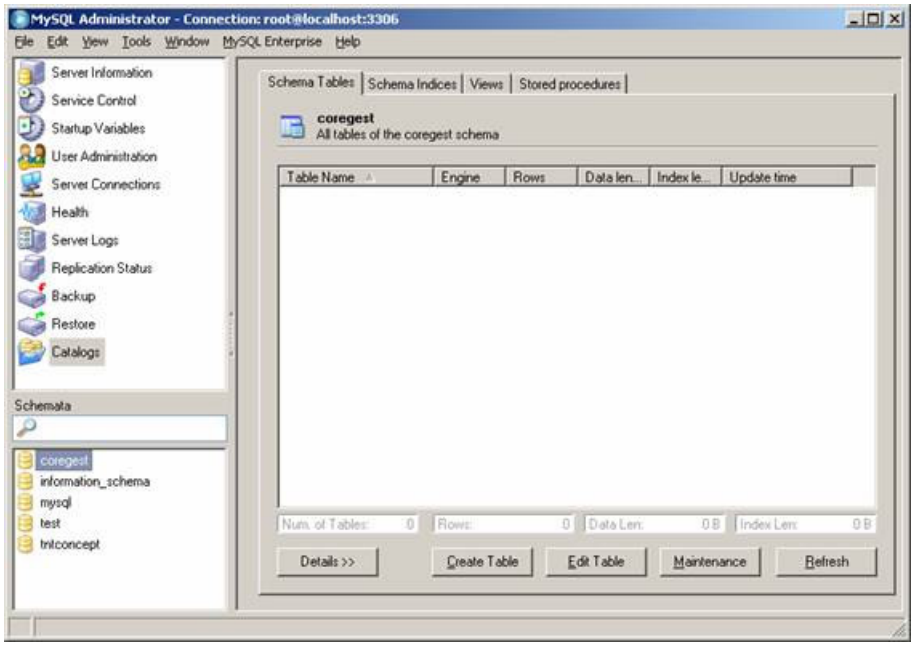
Pulsamos OK



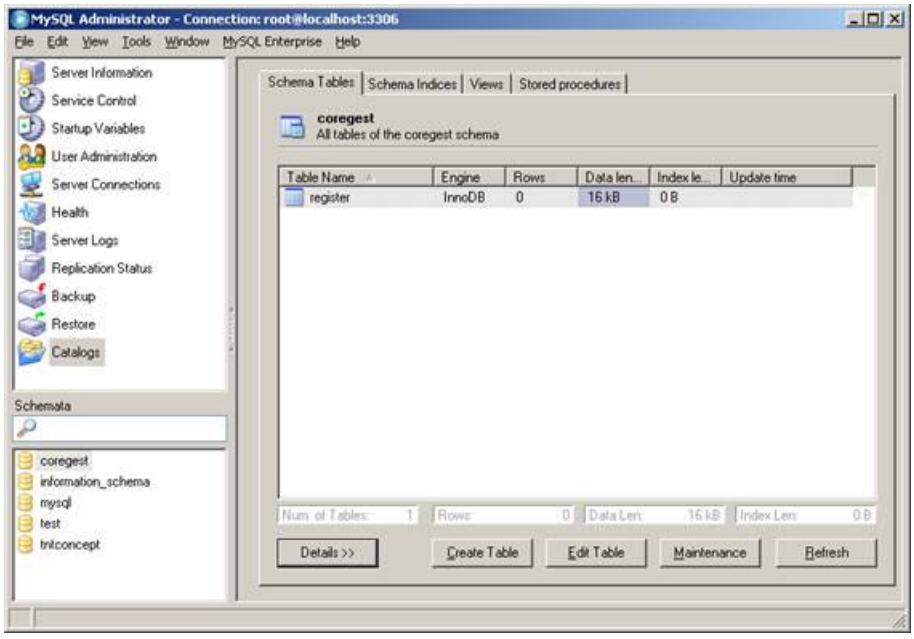
Pulsando seobre el fondo con el botón secundario elegimos "Create a new Schema"



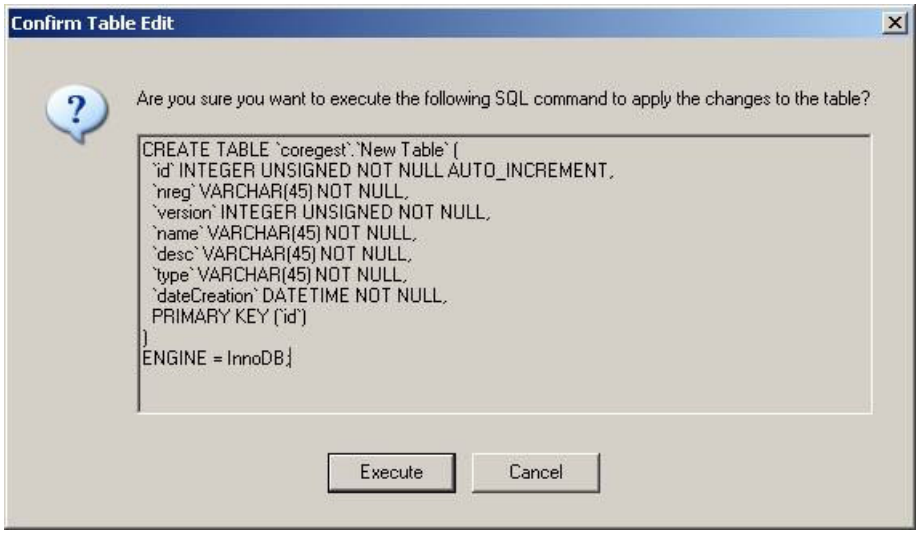
Pulsamos OK



Ahora hay que crear las tablas. Vamos a usar las tablas descritas en el artículo del gestor documental, con alguna variación



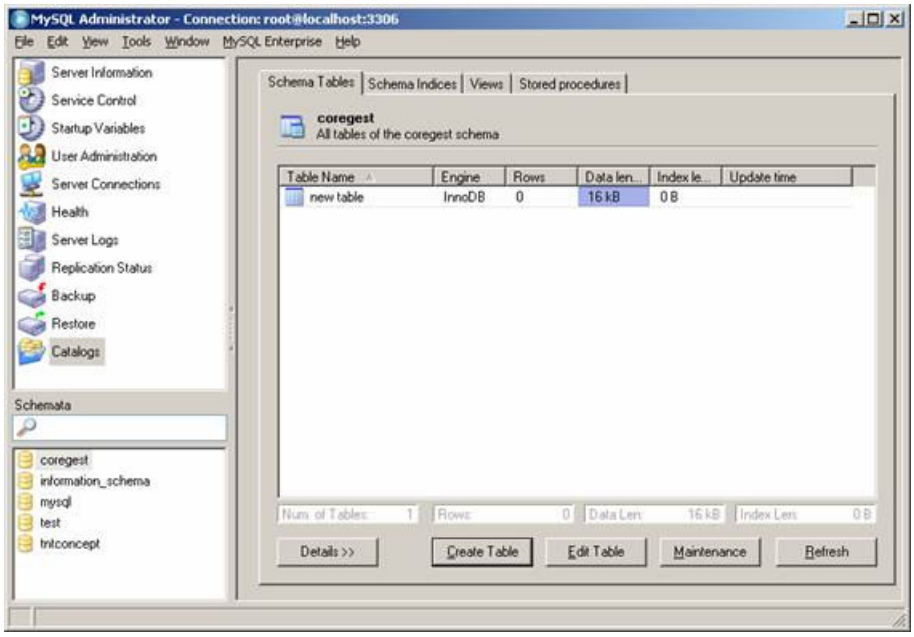
Pulsamos en "Apply changes"



Y nos muestra el SQL necesario para generar la tabla.

```
CREATE TABLE `coregest`.`register` (
  `id` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  `nreg` VARCHAR(45) NOT NULL,
  `version` INTEGER UNSIGNED NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `desc` VARCHAR(45) NOT NULL,
  `type` VARCHAR(45) NOT NULL,
  `dateCreation` DATETIME NOT NULL,
  PRIMARY KEY (`id`)
)
ENGINE = InnoDB;
```

Cerramos la ventana y nos muestra la tabla creada



Procederemos de igual manera con el resto de las tablas.

Añadir un módulo estándar al proyecto: hibernate, driver de mysql, etc.

Lo primero que debemos hacer es localizar el módulo que queremos añadir. Vamos a comenzar por el driver de MySql. Le echamos un vistazo a la página

http://www.ibiblio.org/maven/

y vemos que nos muestra una estructura de carpetas muy amplia. Localizamos la carpeta mysql y vemos que en ella hay varias carpetas: jars, licenses y poms. Dentro de jars está la carpeta mysql-connector-java-5.0.5 (hay más, pero me quedo con la última versión del driver) y hay en la carpeta poms el fichero mysql-connector-5.0.5.pom. Ya tengo toda la información que necesito:

- groupId = mysql (es el nombre de la carpeta desde ibiblio.org/maven)
- artifactId= mysql-connector-java (el nonbre del jar sin la versión ni extensión)
- version=5.0.5 (el que hayamos elegido)

Investigando un poco en la carpeta maven veremos el hibernate y otros módulos interesantes (hay cientos…). Algunas carpetas tienen nombres largos tipo net.sf.jaspereports, que habrá que colocar convenientemente en el groupId.

Añadir el driver de MySql al proyecto

Para añadir el driver de MySql lo vamos a hacer siguiendo el estándar de Maven. Para ello modificamos el POM del módulo coregest-core, añadiéndole las dependencias pertinentes:

Coregest-core\pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>net.sf.coregest</groupId>

  <artifactId>coregest-core</artifactId>

  <packaging>jar</packaging>

  <version>1.0-SNAPSHOT</version>

  <name>coregest-core</name>
```

```
  <url>http://maven.apache.org</url>
  <dependencies>
        <dependency>
              <groupId>mysql </groupId>
              <artifactId>mysql-connector-java</artifactId>
              <version>5.0.5</version>
              <scope>compile</scope>
        </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Probamos a instalar el módulo coregest-core.

Automáticamente se descargará e instalará en el repositorio local el jar del driver de mysql:

Como vemos, el repositorio local se va poblando poco a poco con los módulos necesarios.

Ahora instalamos el paquete coregest (la aplicación web) para ver si se copia también el driver en la aplicación

En la carpeta coregest\target\coregest\web-inf\lib vemos que no lo copia, luego si necesitamos incluirlo en el paquete war hay que añadirlo a las dependencias del segundo módulo también. Esto dependerá de si queremos incluir el driver en el war o no (por ejemplo, por motivos de licencias). También se puede añadir en un pom del que hereden ambos proyectos, que es un mecanismo diseñado en Maven para proyectos con muchos módulos. Si queremos añadir el driver al war añadimos la dependencia al coregest\pom.xml.

Añadir las dependencias de Hibernate al proyecto

Para añadir hibernate al proyecto hacemos lo mismo. Esta vez, buscando en ibiblio obtenemos (si usamos anotaciones son dos dependencias):

- groupId = org.hibernate (el groupId hibernate tiene versiones más antiguas)
- artifactId = hibernate, hibernate-annotations (si vamos a usar anotaciones)
- version = 3.2.0.ga (hay varias combinaciones posibles, ver hibernate.org). Elegimos una.

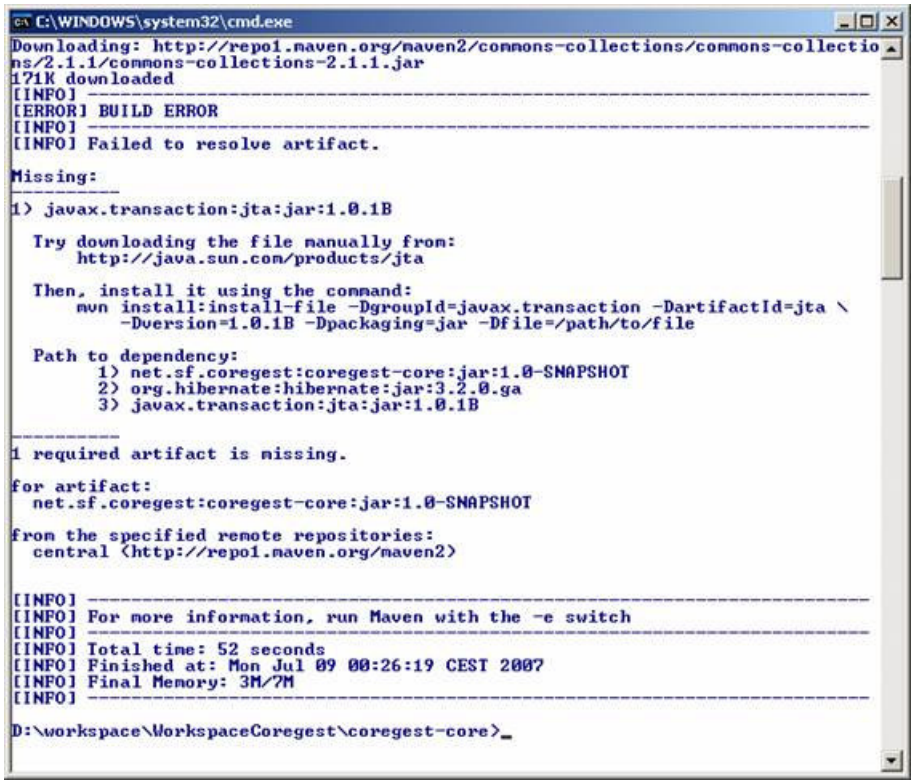
Nos quedan así los pom de los módulos:

```
....
    <dependency>
            <groupId>org.hibernate</groupId>
```

```
                <artifactId>hibernate</artifactId>
                <version>3.2.0.ga</version>
                <scope>compile</scope>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-annotations</artifactId>
                <version>3.2.0.ga</version>
                <scope>compile</scope>
        </dependency>
...
```

También hay que añadirlas al coregest ya que nos interesa que en el war vayan los jar de hibernate.

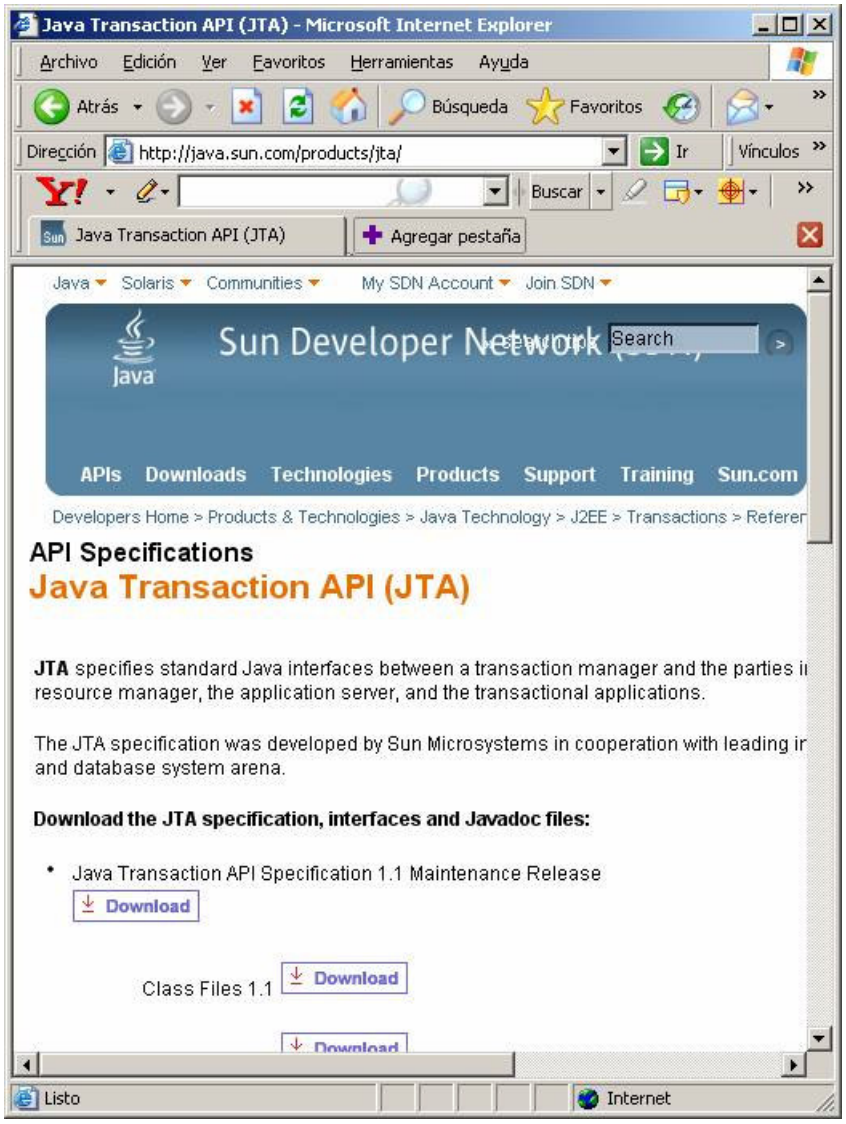
Si ahora hacemos un mvn install del proyecto coregest-core nos sale:



Esto se debe a que por motivos de licencia no está en ibiblio el módulo jta de sun.com. Hay que instalarlo a mano. Siguiendo las instrucciones en pantalla, localizamos el jar en sun.com. Probablemente las limitaciones de las licencias de Sun impiden la inclusión de estas bibliotecas en ibiblio.org.

Instalación manual del Sun JTA

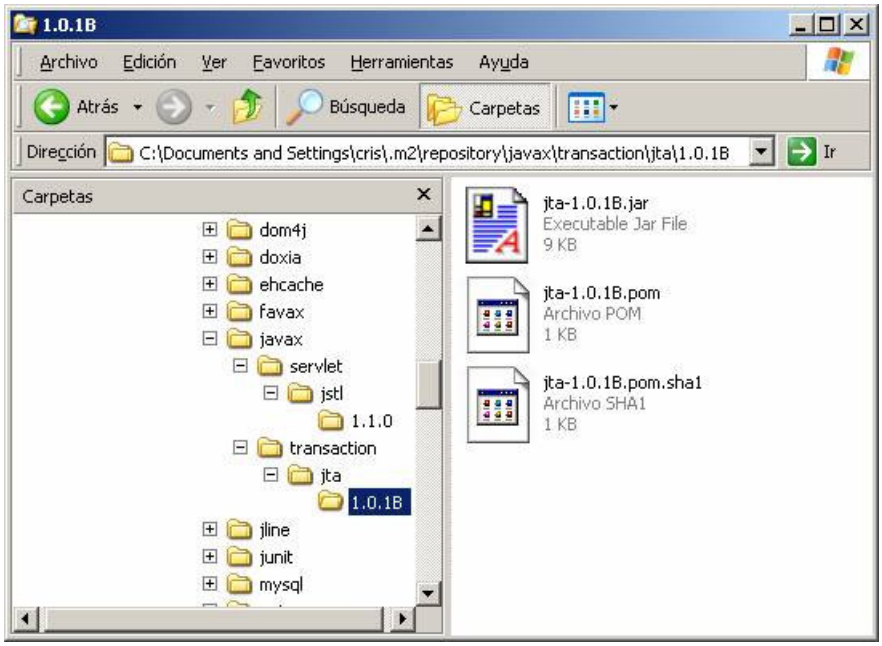
Nos conectamos a sun.com y localizamos el zip de JTA 1.0.1B



Descargamos el fichero jta-1_0_1B-classes.zip a una carpeta local, en mi caso d:\apps y lo instalamos en nuestro repositorio local con

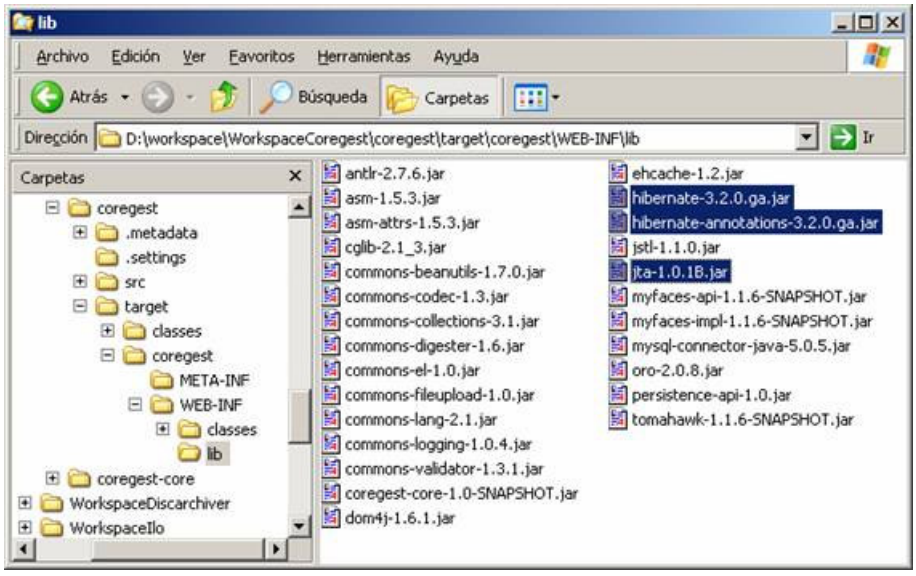
```
mvn install:install-file -DgroupId=javax.transaction -DartifactId=jta
  -Dversion=1.0.1B -Dpackaging=jar -Dfile=d:/apps/ jta-1_0_1B-classes.zip
```

Y se nos instala en local, como podemos comprobar en nuestro repositorio local



Observamos que además lo ha convertido en un jar.

Ahora ya podemos hacer un mvn install sobre los dos módulos y comprobar que todo ha ido bien. El war debe contener los módulos de hibernate:

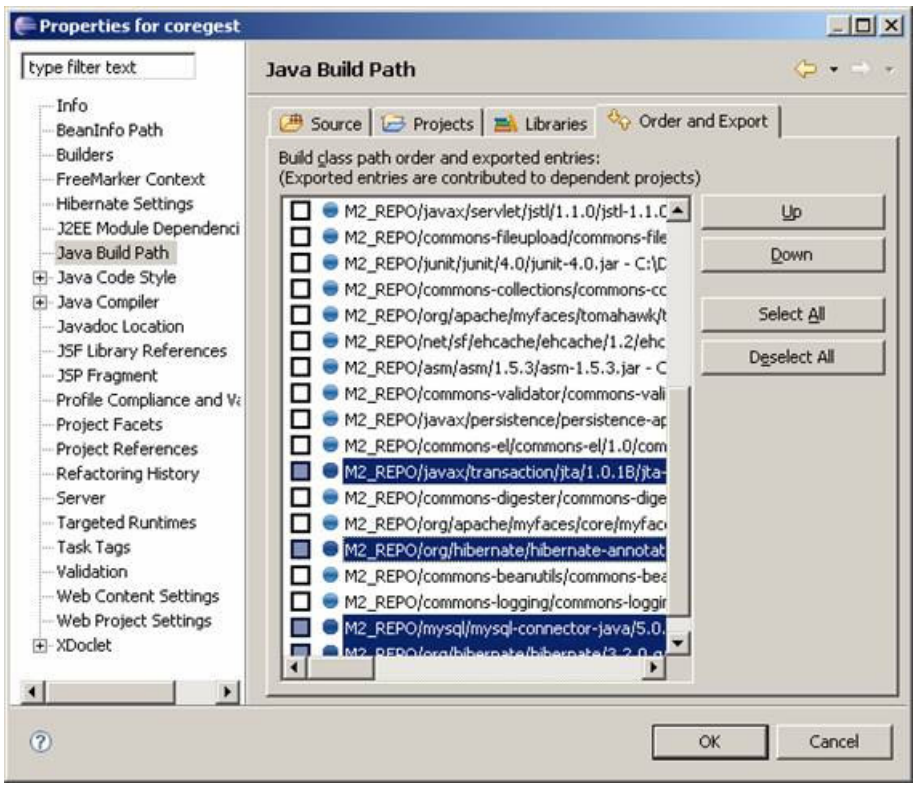


Esto va teniendo buena pinta.

Añadiendo los nuevos módulos a Eclipse

Ya que hemos añadido varios jar al proyecto, hay que actualizar la configuración de los proyectos en Eclipse. Esto lo haremos con mvn eclipse:eclipse (con el Eclipse cerrado).

Y se nos añaden al "Java build path" del Eclipse. Abrimos el Eclipse

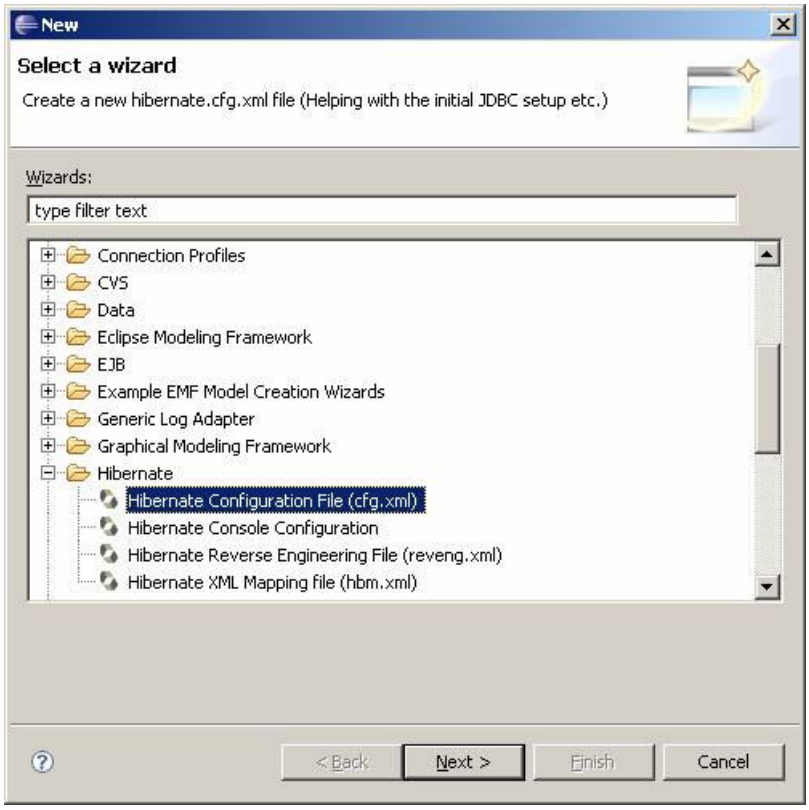


Procedemos de igual manera con coregest-core. Como veis, usar Maven facilita mucho las cosas.

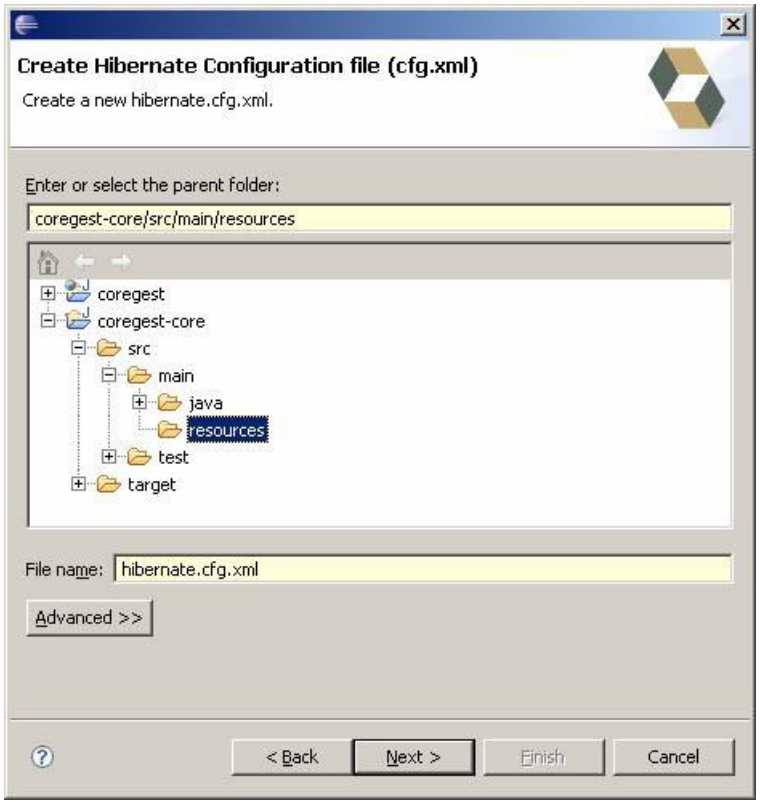
Creación del fichero de configuración de Hibernate

Ahora vamos a introducir Hibernate en la aplicación. Ya que hemos dividido la aplicación separando la lógica de negocio en un módulo separado, será en este módulo en el que añadiremos la configuración de Hibernate.

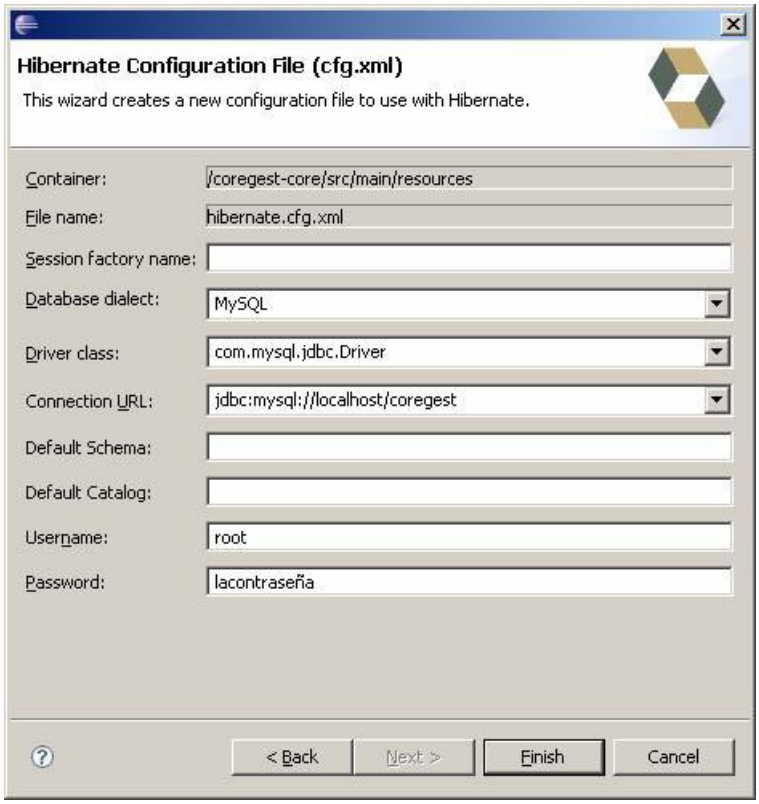
Vamos a comenzar por una configuración simple, siguiendo el tutorial de Alejando Pérez sobre las Hibernate Tools. Lo primero es crear un fichero de configuración de Hibernate, en la carpeta resources de nuestro proyecto (si no exites, creamos coregest-core/src/main/resources). Pulsamos sobre el proyecto coregest-core y creamos un nuevo ficero de configuración con File\New… \Other..\Hibernate configuration file (cfg.xml)



Pulsamos "Next"



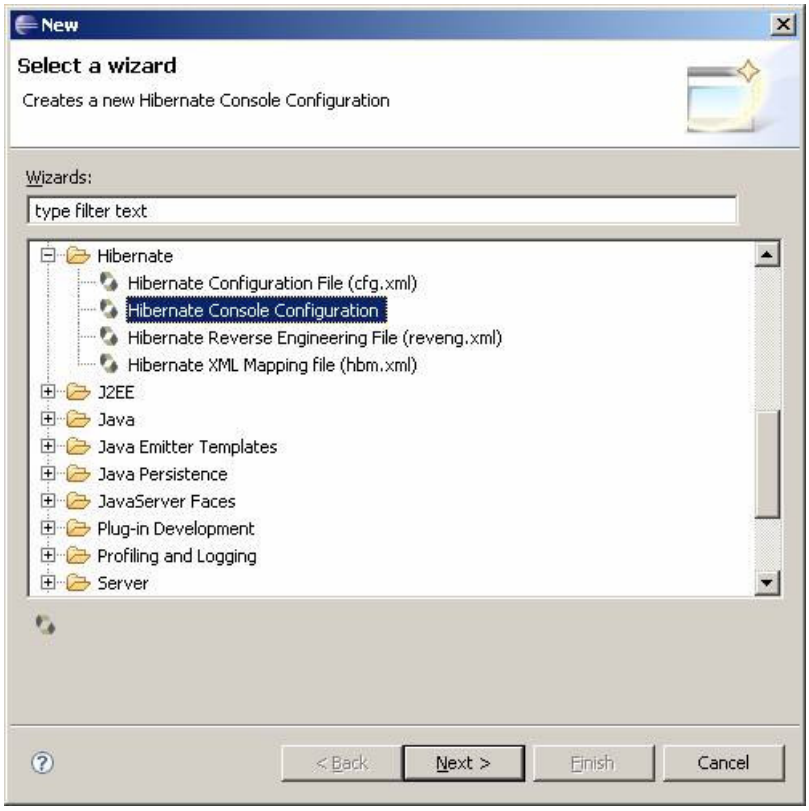
Ahora pulsamos "Next"



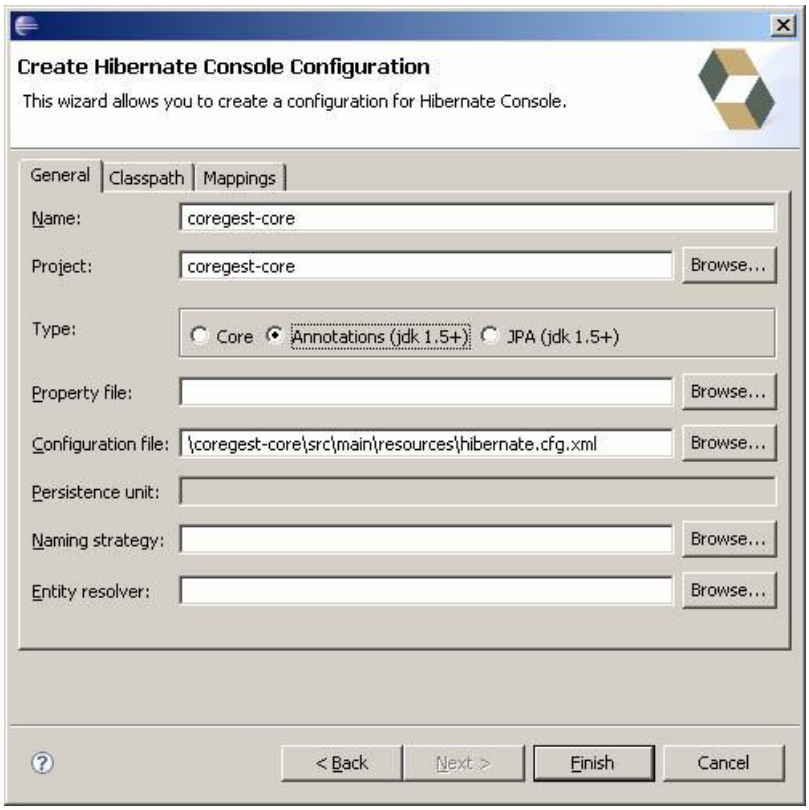
Rellenamos los campos adecuadamente y pulsamos "Finish"

Añadiendo un panel de control de las hibernate tools

En este paso vamos a añadir un panel de control a nuestro módulo coregest-core, de manera que podamos utilizar las hibernate tools sobre él. Pulsamos en "File\New\Others...\Hibernate\Hibernate console configuration"

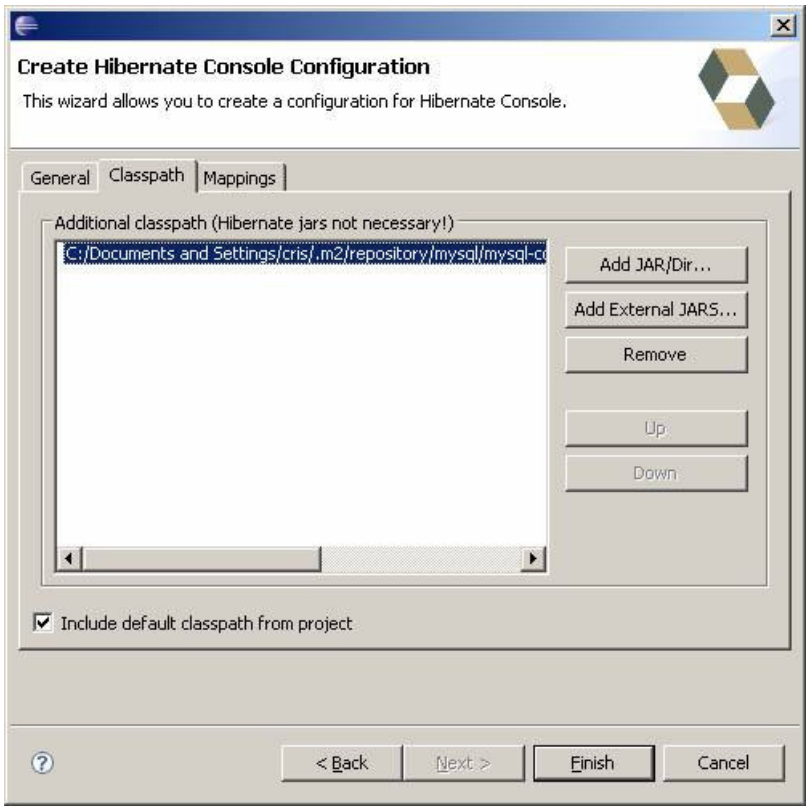


Pulsamos "Next"



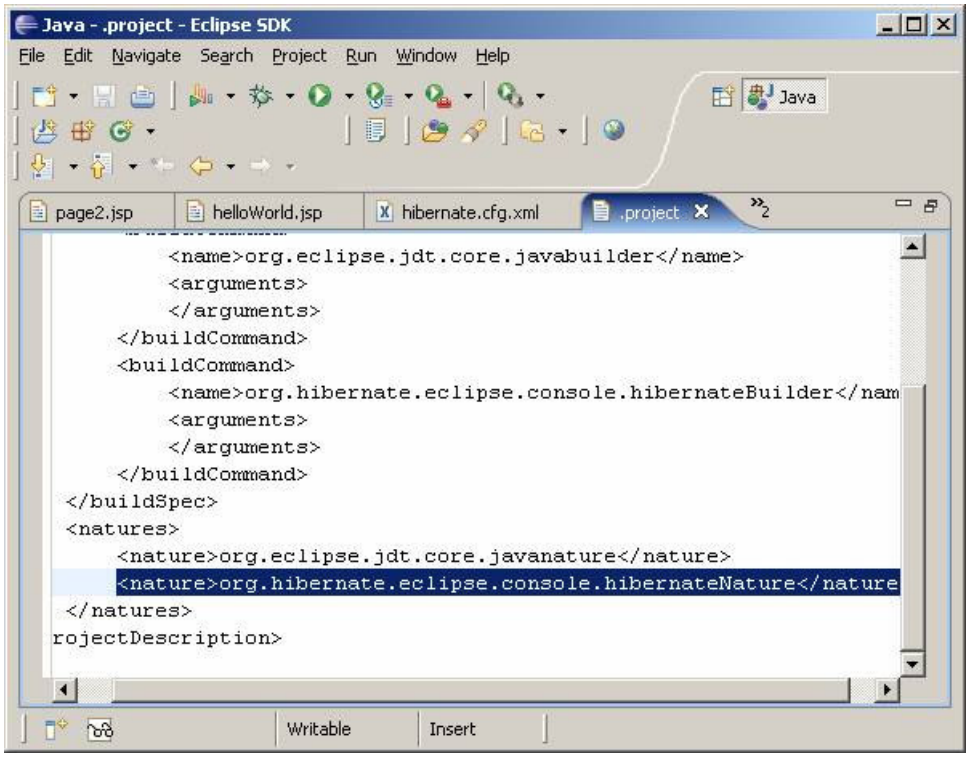
Voy a usar anotaciones, por lo que se lo indico a la consola.

Ahora hay que añadir la ruta al driver de mysql, para que las hibernate tools puedan examinar la base de datos. Pulsamos en la pestaña "Classpath"



He añadido el driver de mysql que tengo en el repositorio local, con el botón "Add externa JAR". También lo podría añadir del target del módulo coregest si lo he incluido en el war. Nos sale un mensaje de aviso.

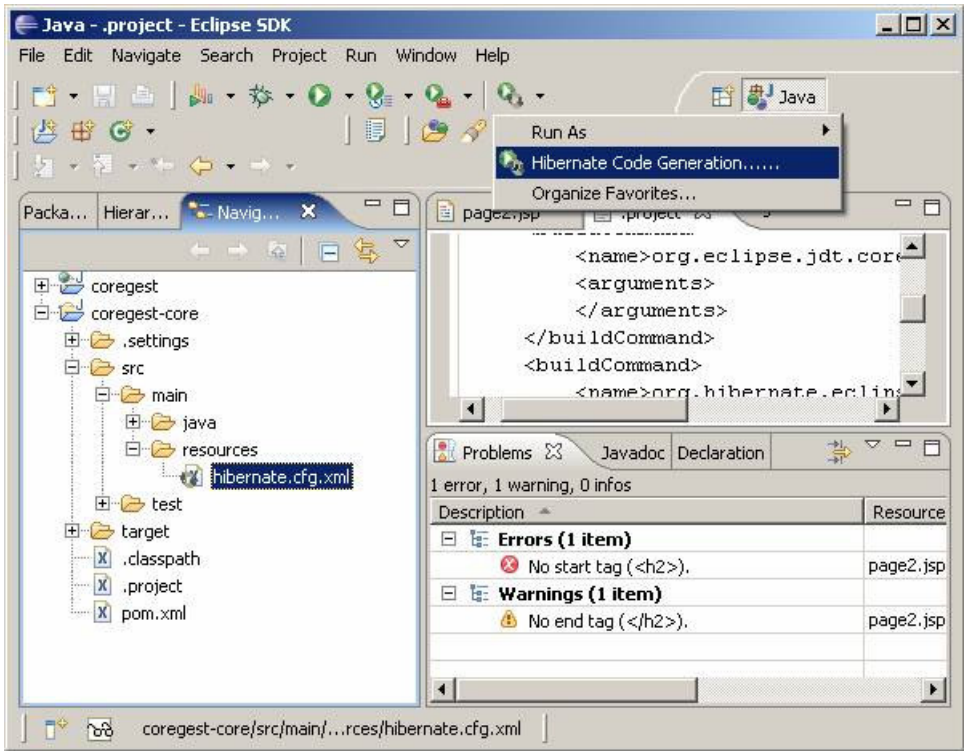
Pulsamo "Yes". Creo que lo que hará será añadirle el nature de hibernate al proyecto. Ummm, veamos:



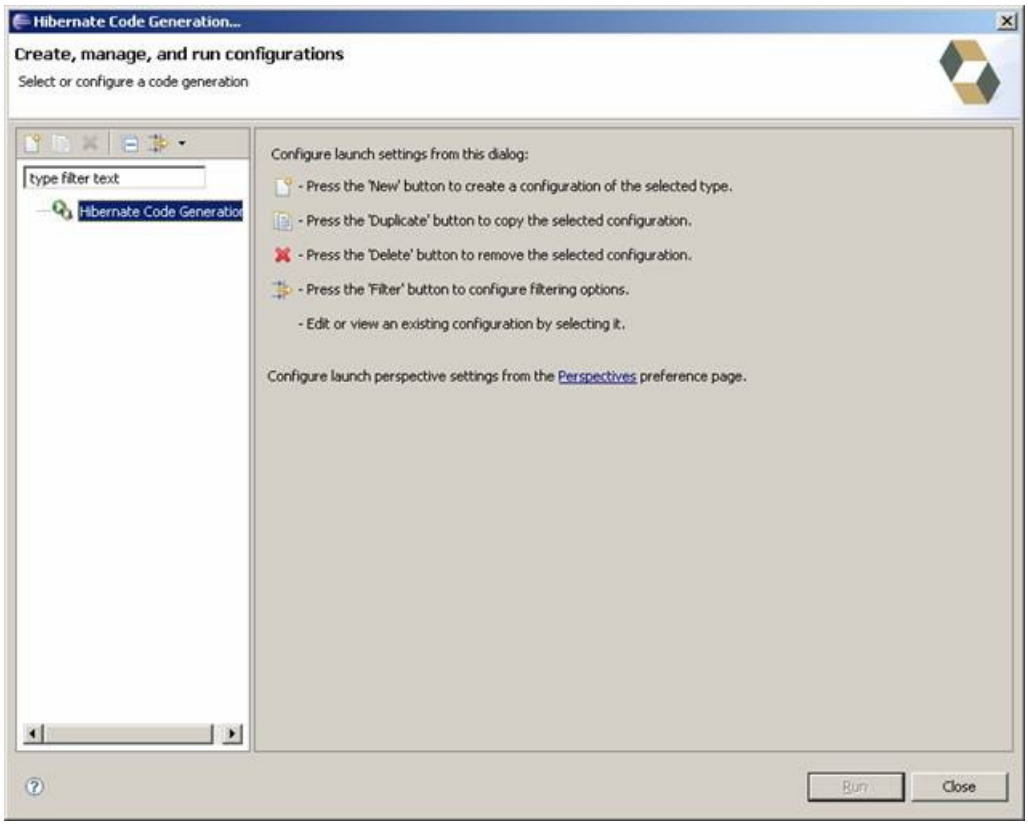
Efectivamente, según sospechabamos le ha añadido el nature de hibernate.

Creando las clases java con las hibernate tools

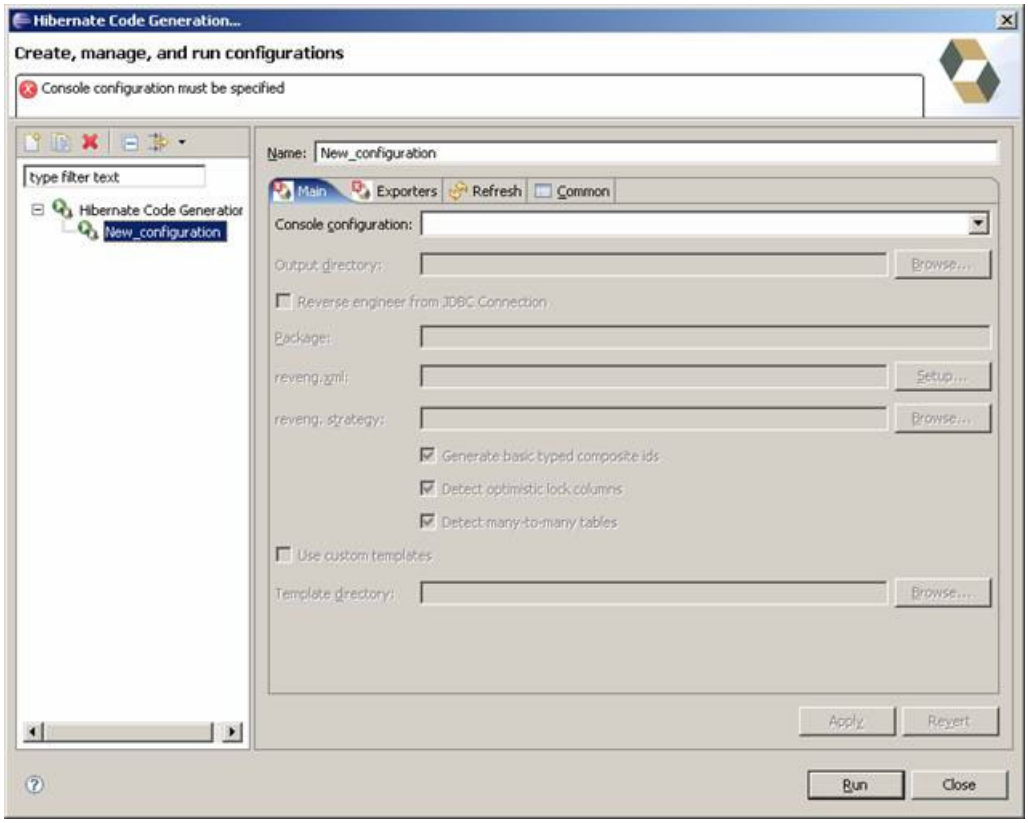
Buscamos el icono que permite lanzar el asistente de genearción de código de las hibernate tools



Es pequeñito, cuesta verlo.

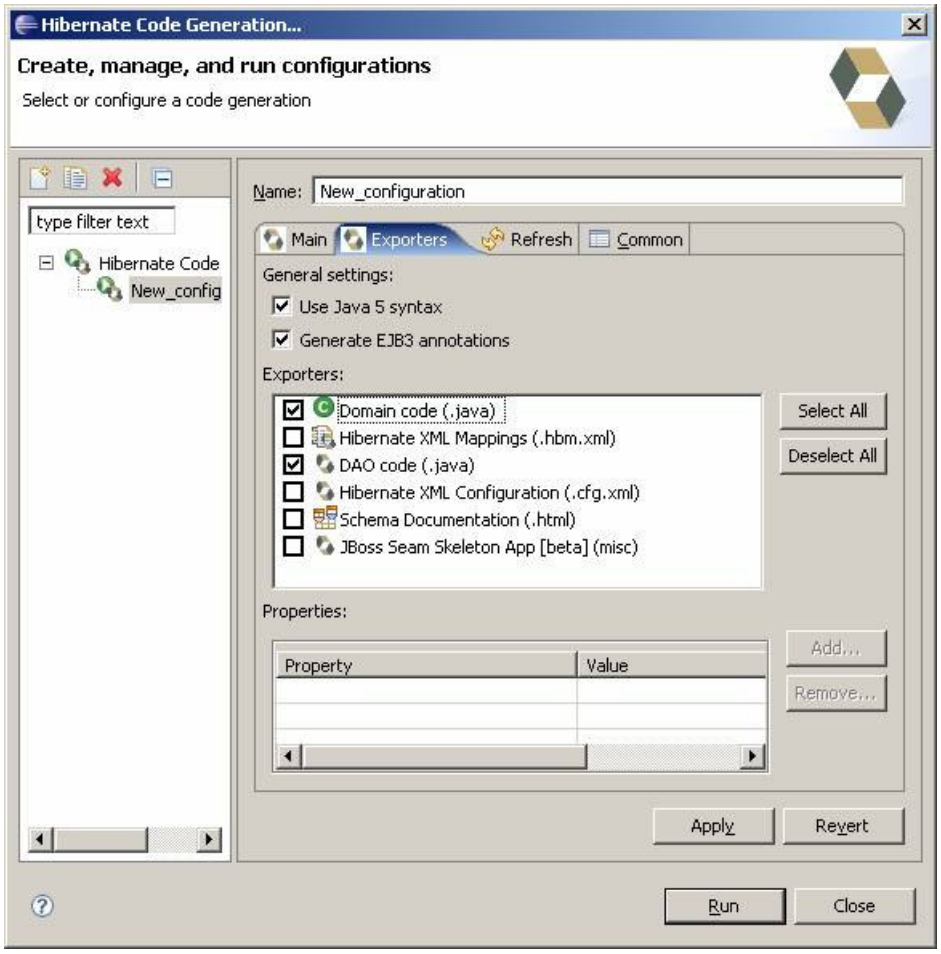


Pulsamos con doble clic sobre "Hibernate code configuration"



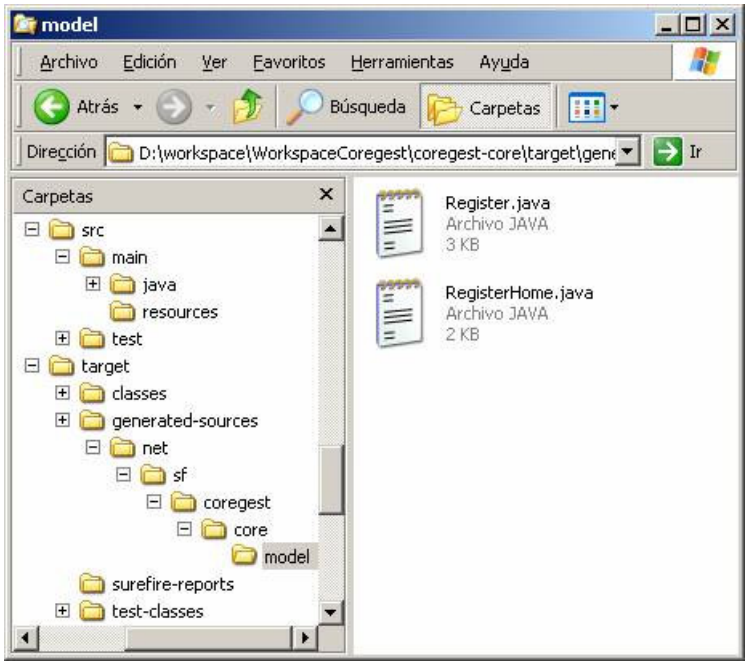
Ahora rellenamos los campos (para más información, leer el tutorial de hibernate tools de Alejandro Pérez http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateTools

Ahora nos vamos a la pestaña de "Exporters"



En nuestro caso queremos utilizar las anotaciones de Java 5.0.

Pulsamos "Run". No olvidar crear la carpeta generated-sources y hacer un refresh (F5) del módulo coregest-core antes de ejecutar el "Run".



Como vemos ha creado un par de archivos para la tabla register. El texto del fichero generado es simple (nos interesa el register.java)

Register.java

```
import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;



/**

 * Register generated by hbm2java

 */

@Entity

@Table(name = "register", catalog = "coregest")

public class Register implements java.io.Serializable {



	private int id;



	private int version;



	private String nreg;



	private String name;



	private String desc;



	private String type;



	private Date dateCreation;



	public Register() {

	}



	public Register(int id, String nreg, String name, String desc, String type,

			Date dateCreation) {

		this.id = id;

		this.nreg = nreg;

		this.name = name;

		this.desc = desc;

		this.type = type;

		this.dateCreation = dateCreation;

	}
```

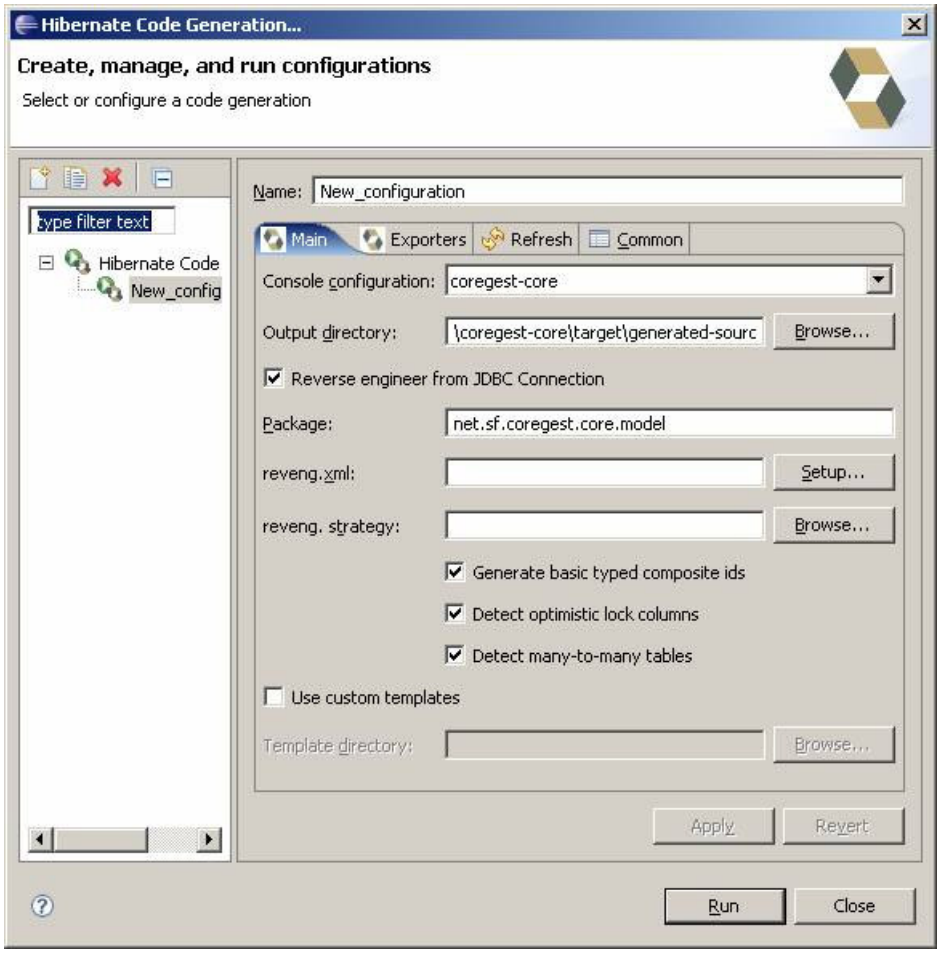
```
@Id
@Column(name = "id", unique = true, nullable = false)
public int getId() {
        return this.id;
}

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

@Column(name = "version", nullable = false)
public int getVersion() {
        return this.version;
}

public void setVersion(int version) {
        this.version = version;
}

@Column(name = "nreg", nullable = false, length = 45)
public String getNreg() {
        return this.nreg;
}

public void setNreg(String nreg) {
        this.nreg = nreg;
}

@Column(name = "name", nullable = false, length = 45)
public String getName() {
        return this.name;
}

public void setName(String name) {
        this.name = name;
}

@Column(name = "desc", nullable = false, length = 45)
public String getDesc() {
```

```
		return this.desc;

	}


	public void setDesc(String desc) {

		this.desc = desc;

	}


	@Column(name = "type", nullable = false, length = 45)

	public String getType() {

		return this.type;

	}


	public void setType(String type) {

		this.type = type;

	}


	@Column(name = "dateCreation", nullable = false, length = 0)

	public Date getDateCreation() {

		return this.dateCreation;

	}


	public void setDateCreation(Date dateCreation) {

		this.dateCreation = dateCreation;

	}


}
```

Podríamos simplificarlo, tal y como nos indica Alejandro.

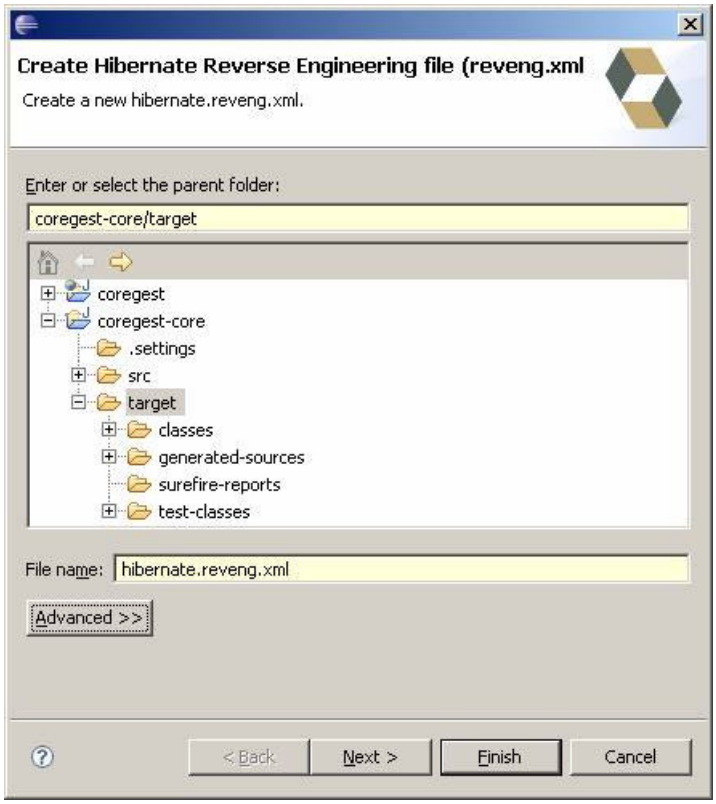
Refinando la generación de las clases con el fichero revenge.xml

Ahora vamos a crear el fichero revenge.xml tal y como se indica en el tutorial de las hibernate tools. Pulsamos en "Hibernate code generation" y en la pantalla de creación del código pulsamos el botón "Setup" del campo "revenge.xml"

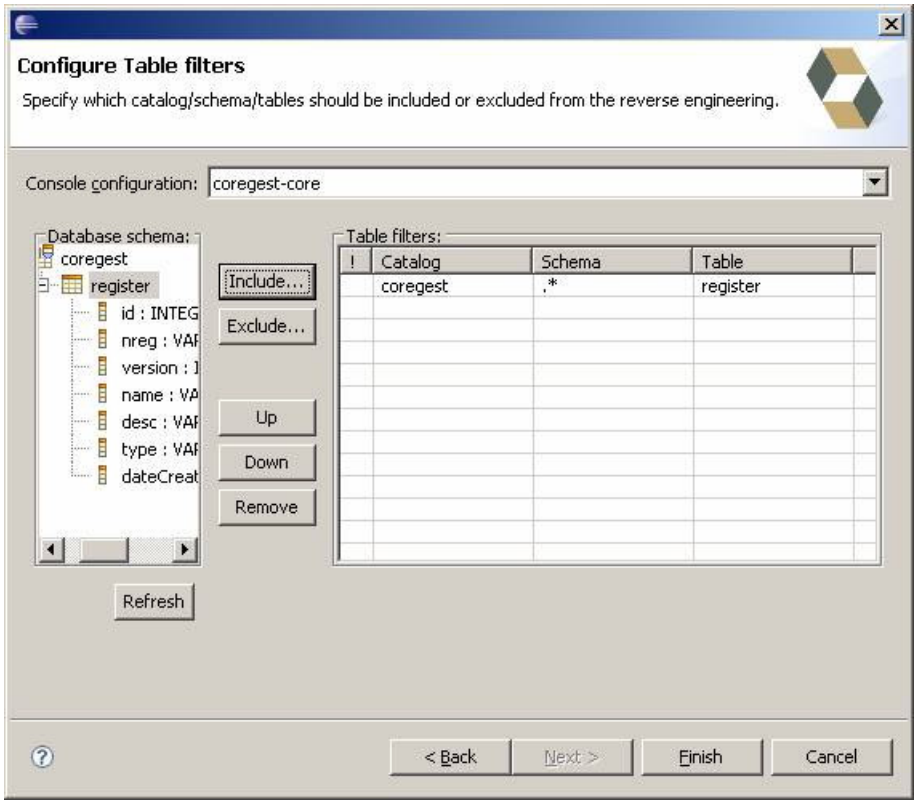


Al pulsar el botón nos pregunta si deseamos crear un nuevo fichero

Pulsamos "Create new"

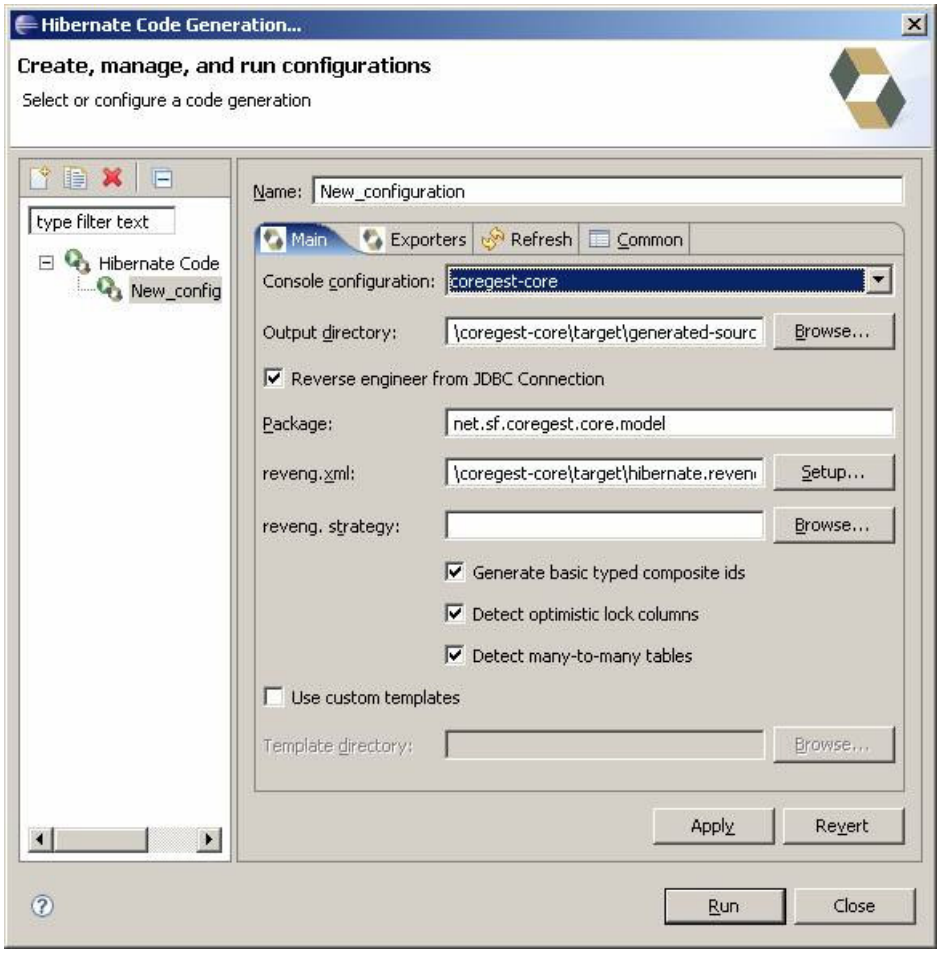


Seleccionamos la ruta y pulsamos "Next"



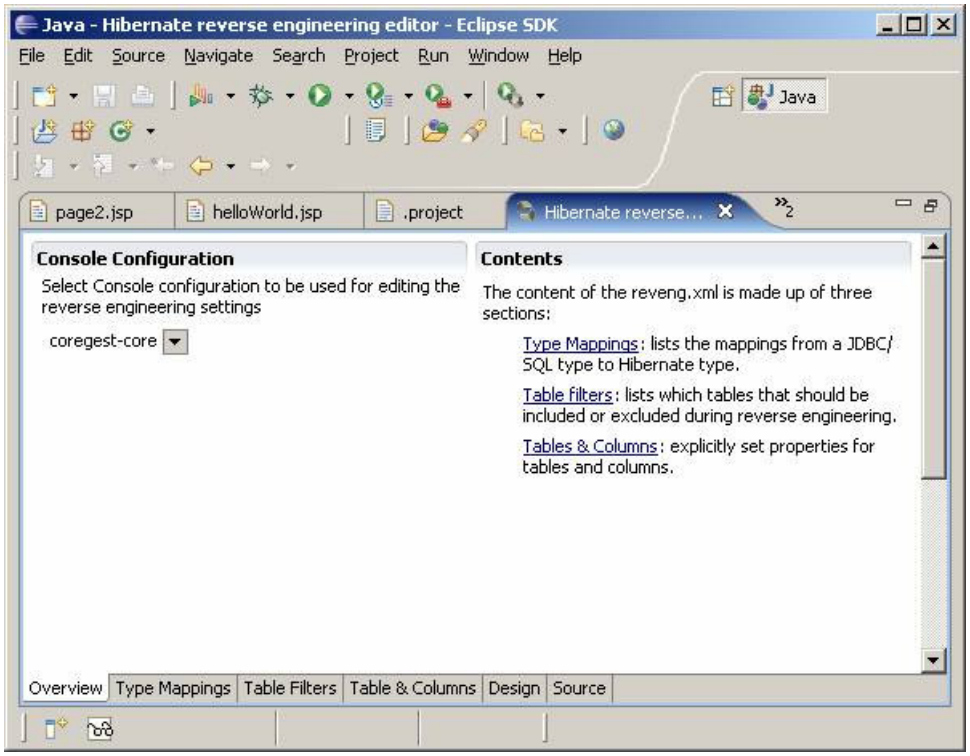
Al abrirse este cuadro de diálogo por primera vez se mostrará en blanco. Pulsamos "Refresh" y se cargan las bases de datos existentes.

Elegimos las tablas que queremos incluir en la generación de código y pulsamos "Finish"

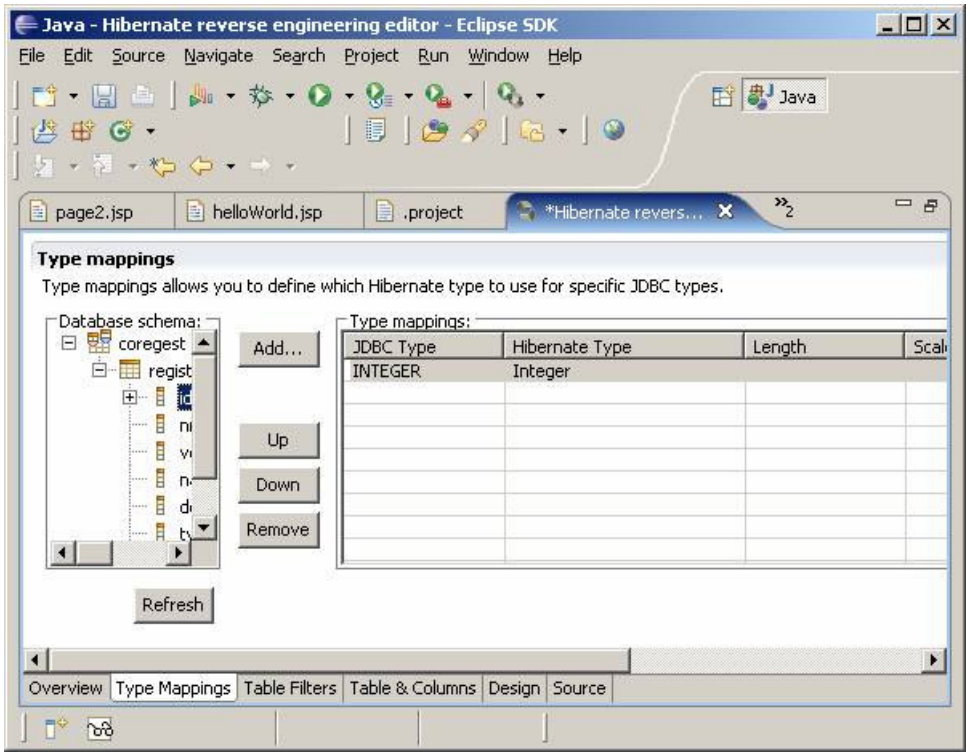


Ahora ya aparece la configuración de revenge.xml. Pulsamos "Close"

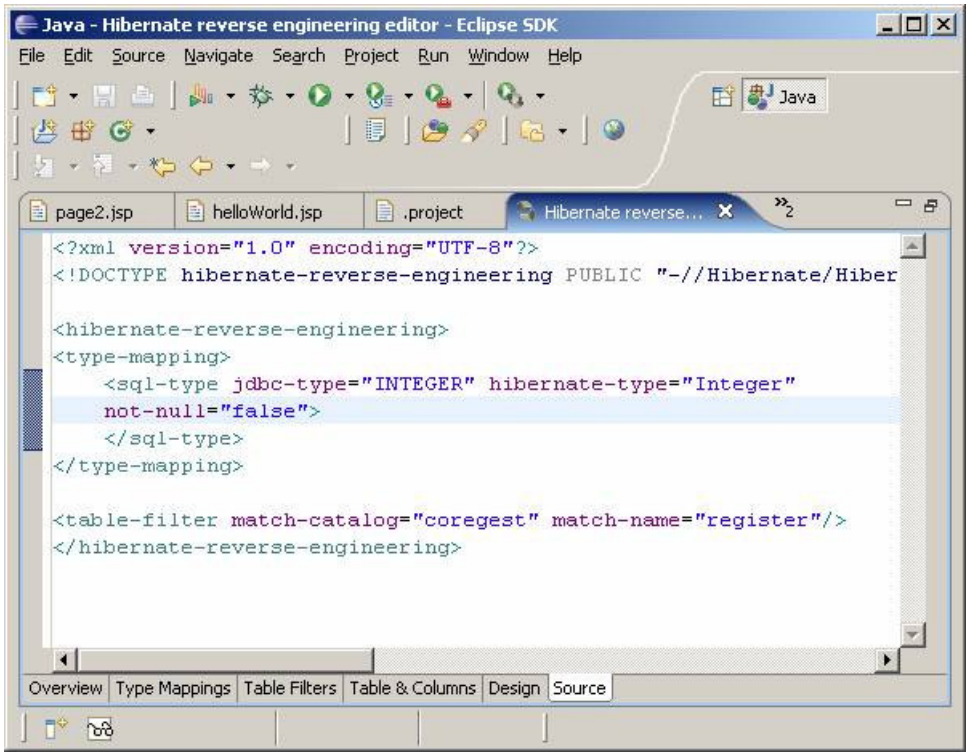
Guardamos los cambios y volvemos a la pantalla principal de Eclipse. Ahora vamos a buscar el fichero hibernate.revenge.xml en Eclipse y lo abrimos. El editor predeterminado para este fichero es específico.



Nos vamos a la pestaña "Type mappings" para cambiar los mapeos de tipos de datos.

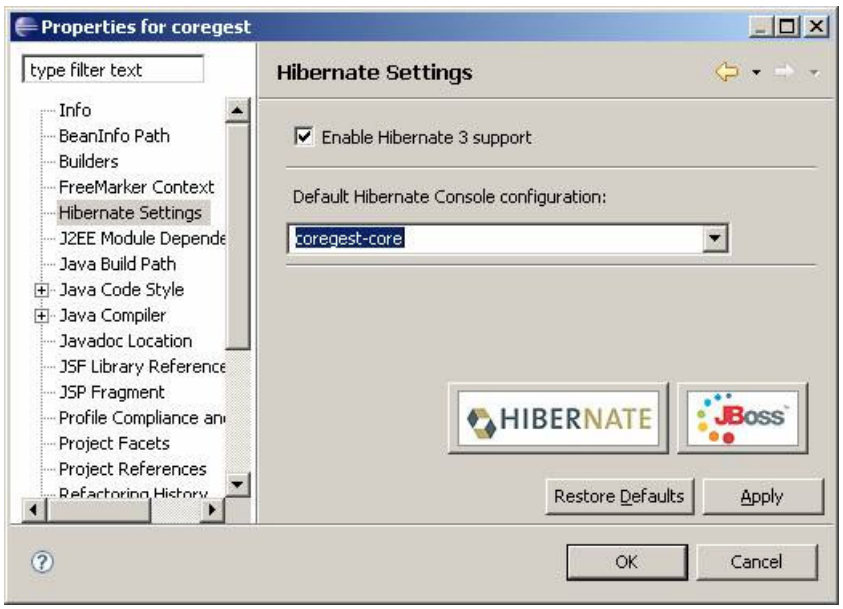


Pulsamos el botón "Refresh" y se cargan los datos de las tablas. Pulsamos en el campo "id" de la primera tabla y luego el botón "Add...". Nos añade un mapeo de INTEGER a "int", que no nos interesa. Lo cambiamos por "Integer", tal y como se ve en la figura. También cambiamos en la última columna el "Not null" a false (para que sea nullable, como nos sugiere Alex). Esto genera un fichero hibernate.revenge.xml que podemos ver en la pestaña source



Añadiendo el soporte para Hibernate 3 en el proyecto coregest-core

Abrimos las propiedades del proyecto coregest-core y seleccionamos la opción Hibernate



Activamos el soporte para Hibernate 3 y seleccionamos la consola que debe usar Eclipse

Añadiendo la persistencia a nuestra aplicación sobre el módulo de la lógica de negocio

En este apartado vamos a escribir los elementos básicos para que nuestra aplicación sea capaz de persistir en la base de datos un objeto, configurando el Hibernate y haciendo pruebas de su configuración.

Creando la configuración de Hibernate para las pruebas unitarias

Para las pruebas unitarias he creado el siguiente fichero de configuración, que he dejado en la carpeta coregest-core\src\test\resources

coregest-core\src\test\resources\hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration SYSTEM

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

       <session-factory>

             <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>

             <property
name="hibernate.connection.url">jdbc:mysql://localhost/coregest</property>

             <property name="hibernate.connection.username">root</property>

              <property name="hibernate.connection.password">sesamo</property>

             <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>



             <property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>

          <!--  thread is the short name for
org.hibernate.context.ThreadLocalSessionContext

                    and let Hibernate bind the session automatically to the thread

                    Sinde Hibernate 3.1

             -->



             <property name="hibernate.current_session_context_class">thread</property>




             <!-- Use the C3P0 connection pool provider -->

             <!--

             <property name="hibernate.c3p0.min_size">5</property>

             <property name="hibernate.c3p0.max_size">20</property>

             <property name="hibernate.c3p0.timeout">300</property>

              <property name="hibernate.c3p0.max_statements">50</property>

             <property name="hibernate.c3p0.idle_test_period">3000</property>

             -->



             <!-- Show and print nice SQL on stdout -->

             <property name="show_sql">true</property>

             <property name="format_sql">true</property>



             <!-- List of XML mapping files -->

             <mapping class="net.sf.coregest.core.model.Register"/>

       </session-factory>
```

```
</hibernate-configuration>
```

Para las pruebas unitarias no voy a usar C3PO (el pool de conexiones) ya que en la configuración actual del proyecto no se incluye por defecto. Creo que el entorno de pruebas de Maven sólo pone a disposición de la aplicación de pruebas unitarias los jar que aparecen explícitamente en las dependencias, y C3PO no está incluido. Nos lo apuntamos como deberes...

Además le vamos a añadir un fichero de logs mínimo

coregest-core\src\test\resources\logs4j.properties

```
# Direct log messages to stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE}?%5p %c{1}:%
L - %m%n

# Root logger option

log4j.rootLogger=INFO, stdout

# Hibernate logging options (INFO only shows startup messages)

log4j.logger.org.hibernate=INFO

# Log JDBC bind parameter runtime arguments

log4j.logger.org.hibernate.type=INFO
```

Creando la inicialización de Hibernate con nuestro HibernateUtil

Para que podamos obtener una sesión de Hibernate se suele crear una clase auxiliar llamada HibernateUtil, de la que hay muchos ejemplos en Internet. Vamos a comenzar usando una versión simplificada y más adelante se podrá sustituir por una versión threadsafe (segura en multihilos) y con soporte para JNDI.

/coregest-core/src/main/java/net/sf/coregest/persistence/HibernateUtil.java

```
package net.sf.coregest.persistence;

import org.hibernate.*;

import org.hibernate.cfg.*;



public class HibernateUtil {

        private static SessionFactory sessionFactory;



        static {

                try {

                        System.out.println("HibernateUtil - constructor begin");

                        sessionFactory=new AnnotationConfiguration().configure()
```

```
                                .buildSessionFactory();
                } catch (Throwable ex) {
                        System.out.println("HibernateUtil - constructor error");
                        ex.printStackTrace();
                        throw new ExceptionInInitializerError(ex);
                }
                System.out.println("HibernateUtil - constructor end");
        }

        public static SessionFactory getSessionFactory() {
                // Alternatively, you could look up in JNDI here

                return sessionFactory;
        }

        public static void shutdown() {
                // Close caches and connection pools
                getSessionFactory().close();
        }
}
```

Para esta prueba hemos usado la clase que viene en el libro "Java Persistence with Hibernate" (ver enlaces)

Creando la primera clase a persistir: register

La clase Register que hemos obtenido con las Hibernate Tools la copiamos a su carpeta. Por conveniencia y para que esté actualizada, la vuelvo a copiar aquí:

/coregest-core/src/main/java/net/sf/coregest/core/model/Register.java

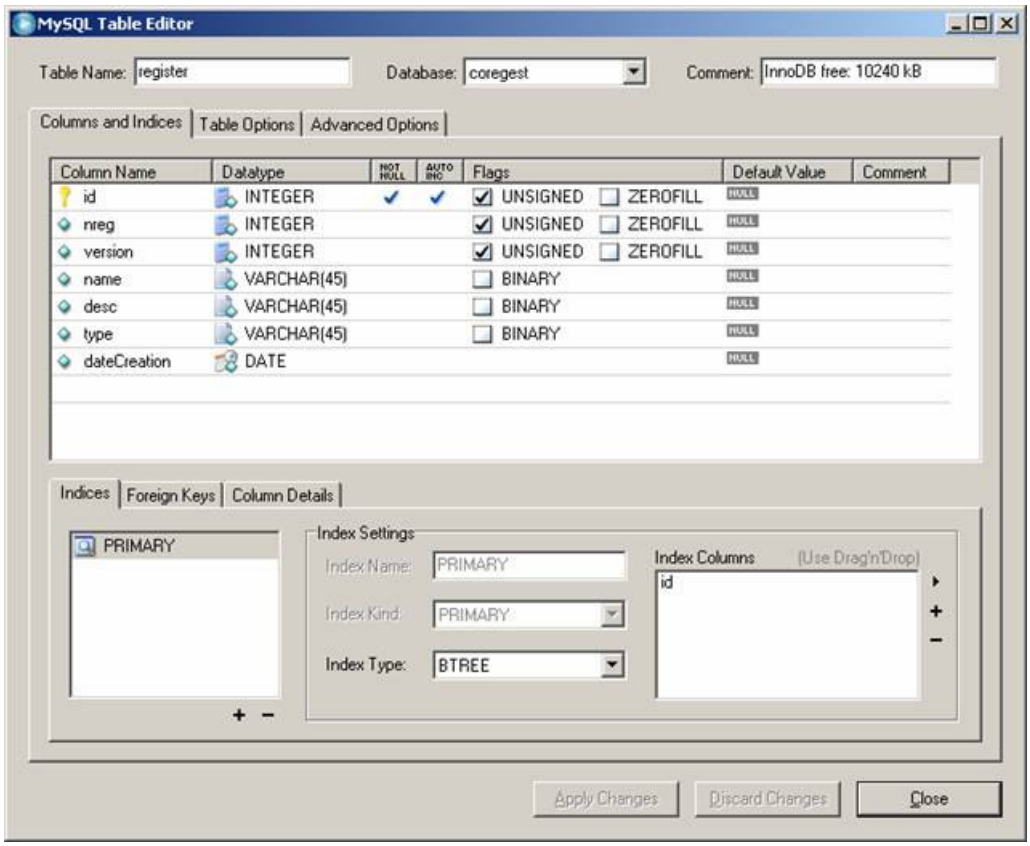
```
package net.sf.coregest.core.model;



// Generated 12-jul-2007 19:03:24 by Hibernate Tools 3.2.0.b9



import java.util.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;



/**

 * Register generated by hbm2java

 */

@Entity
```

```
@Table(name = "register", catalog = "coregest")
public class Register implements java.io.Serializable {

	private long id;

//	private int version;
//
//	private int nreg;
//
	private String name;
//
//	private String desc;
//
//	private String type;

	private Date dateCreation;

	public Register() {
	}

	public Register(long id, int nreg, String name, String desc, String type,
			Date dateCreation) {
		this.id = id;
//		this.nreg = nreg;
//		this.name = name;
//		this.desc = desc;
//		this.type = type;
//		this.dateCreation = dateCreation;
	}

	@Id
	@Column(name = "id", unique = true, nullable = true)
	public Long getId() {
		return this.id;
	}

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

```
//
//	@Column(name = "version", nullable = false)
//	public int getVersion() {
//		return this.version;
//	}
//
//	public void setVersion(int version) {
//		this.version = version;
//	}
//
//	@Column(name = "nreg", nullable = false, length = 45)
//	public int getNreg() {
//		return this.nreg;
//	}
//
//	public void setNreg(int nreg) {
//		this.nreg = nreg;
//	}
//
	@Column(name = "name", nullable = false, length = 45)
	public String getName() {
		return this.name;
	}

	public void setName(String name) {
		this.name = name;
	}
//
//	@Column(name = "desc", nullable = true, length = 45)
//	public String getDesc() {
//		return this.desc;
//	}
//
//	public void setDesc(String desc) {
//		this.desc = desc;
//	}
//
//	@Column(name = "type", nullable = true, length = 45)
//	public String getType() {
```

```
//                return this.type;
//        }
//
//        public void setType(String type) {
//                this.type = type;
//        }

        @Column(name = "dateCreation", nullable = true, length = 0)
        public Date getDateCreation() {
                return this.dateCreation;
        }

        public void setDateCreation(Date dateCreation) {
                this.dateCreation = dateCreation;
        }

}
```

Como veis, para las pruebas he anulado algunos campos que antes había definido. Además hay que configurar la base de datos para que los campos sean nulab.es.



Sólo he puesto como no nulo el campo id.

Pruebas unitariasdel módulo coretest-core usando Hibernate

Ahora nos creamos una clase de pruebas. Nuesto Maven nos ha creado la clase AppTest.java, que vamos a modificar convenientemente:

/coregest-core/src/test/java/net/sf/coregest/AppTest.java

```
package net.sf.coregest;



import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;



import java.util.Date;

import java.util.Iterator;

import java.util.List;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import net.sf.coregest.core.model.*;

import net.sf.coregest.persistence.HibernateUtil;;




/**

 * Unit test for simple App.

 */

public class AppTest

    extends TestCase

{

    /**

     * Create the test case

     *

     * @param testName name of the test case

     */

    public AppTest( String testName )

    {

        super( testName );

    }



    /**

     * @return the suite of tests being tested

     */

    public static Test suite()

    {
```

```
        return new TestSuite( AppTest.class );
    }

    /**
     * Rigourous Test :-)
     */
    /** prueba de listado e inserción de un registro con fecha */
    public void testApp()
    {
        System.out.println("testApp - Beginning");
        listRegisters();
        System.out.println("testApp  fin guardado un registro");
             insertRetister("Registro1", new Date());
             System.out.println("testApp - End");
    }

    /** prueba de inserción de un registro sin fecha y listado */
    public void testApp2() {
       System.out.println("testApp2 - Beginning");
       insertRetister("Registro2", new Date());
       listRegisters();
       System.out.println("testApp2 - End");

    }

    /* clases de utilidad */

    /** Lista todos los registros de tipo Register */
    private void listRegisters(){
       System.out.println("listRegisters - Begins");
       Transaction tx = null;
             Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        System.out.println("listRegisters - Hibernate configured");

        try {
                    tx = session.beginTransaction();
                    List registros = session.createQuery("select r from Register as
r").list();
                    System.out.println("Registros leidos: " + registros.size());
                    for (Iterator iter = registros.iterator(); iter.hasNext();) {
                           Register element = (Register) iter.next();
```

```
                        //log.debug(element);
                        System.out.println(element);


                }
                tx.commit();
            } catch (HibernateException e) {
                e.printStackTrace();
                if (tx != null && tx.isActive())
                    tx.rollback();
            }
            System.out.println("listRegisters - Ends");
    }

    /**  Inserta un registro de tipo Register en la BD */
    private void insertRetister(String name, Date date){

       System.out.println("insertReegister - Beginning");
        Transaction tx = null;
            Session session = HibernateUtil.getSessionFactory().getCurrentSession();
            System.out.println("insertRegister  getted session");

            try {
                tx = session.beginTransaction();
                    Register r = new Register();
                    r.setName(name);
                    r.setDateCreation(date);
                    session.persist(r);
                    System.out.println("insertRegister  guardado el registro " +
name);
                    tx.commit();
                    System.out.println("insertRegister cerrada la tx");

            } catch (HibernateException e) {
                e.printStackTrace();
                if (tx != null && tx.isActive())
                    tx.rollback();
            }
            System.out.println("insertRegister - End");

    }
```

```
}
```

Como vemos sólo hace un par de inserciones en la base de datos coregest, que ahora sólo tiene una tabla: register.

Ejecutando las pruebas unitarias en Maven

Maven ejecuta las pruebas unitarias dentro de la fáse de compilación e instalación del módulo, pero también las podemos llamar con mvn test. Voy a ejecutar las pruebas con la base coregest vacía y capturo el resultado, aunque es largo:

```
D:\workspace\WorkspaceCoregest\coregest-core>mvn test

[INFO] Scanning for projects...

[INFO] ------------------------------------------------------------------------

---

[INFO] Building coregest-core

[INFO]    task-segment: [test]

[INFO] ------------------------------------------------------------------------

---

[INFO] [resources:resources]

[INFO] Using default encoding to copy filtered resources.

[INFO] [compiler:compile]

[INFO] Nothing to compile - all classes are up to date

[INFO] [resources:testResources]

[INFO] Using default encoding to copy filtered resources.

[INFO] [compiler:testCompile]

[INFO] Nothing to compile - all classes are up to date

[INFO] [surefire:test]

[INFO] Surefire report directory: D:\workspace\WorkspaceCoregest\coregest-core\t

arget\surefire-reports



-------------------------------------------------------

 T E S T S

-------------------------------------------------------

Running net.sf.coregest.AppTest

testApp - Beginning

listRegisters - Begins

HibernateUtil - constructor begin

25-jul-2007 23:40:33 org.hibernate.cfg.annotations.Version <clinit>

INFO: Hibernate Annotations 3.2.0.GA

25-jul-2007 23:40:33 org.hibernate.cfg.Environment <clinit>

INFO: Hibernate 3.2.0

25-jul-2007 23:40:33 org.hibernate.cfg.Environment <clinit>
```

```
INFO: hibernate.properties not found
25-jul-2007 23:40:33 org.hibernate.cfg.Environment buildBytecodeProvider
INFO: Bytecode provider name : cglib
25-jul-2007 23:40:33 org.hibernate.cfg.Environment <clinit>
INFO: using JDK 1.4 java.sql.Timestamp handling
25-jul-2007 23:40:33 org.hibernate.cfg.Configuration configure
INFO: configuring from resource: /hibernate.cfg.xml
25-jul-2007 23:40:33 org.hibernate.cfg.Configuration getConfigurationInputStream

INFO: Configuration resource: /hibernate.cfg.xml
25-jul-2007 23:40:34 org.hibernate.cfg.Configuration doConfigure
INFO: Configured SessionFactory: null
25-jul-2007 23:40:34 org.hibernate.cfg.AnnotationBinder bindClass
INFO: Binding entity from annotated class: net.sf.coregest.core.model.Register
25-jul-2007 23:40:34 org.hibernate.cfg.annotations.EntityBinder bindTable
INFO: Bind entity net.sf.coregest.core.model.Register on table register
25-jul-2007 23:40:34 org.hibernate.connection.DriverManagerConnectionProvider co
nfigure
INFO: Using Hibernate built-in connection pool (not for production use!)
25-jul-2007 23:40:34 org.hibernate.connection.DriverManagerConnectionProvider co
nfigure
INFO: Hibernate connection pool size: 20
25-jul-2007 23:40:34 org.hibernate.connection.DriverManagerConnectionProvider co
nfigure
INFO: autocommit mode: false
25-jul-2007 23:40:34 org.hibernate.connection.DriverManagerConnectionProvider co
nfigure
INFO: using driver: com.mysql.jdbc.Driver at URL: jdbc:mysql://localhost/coreges
t
25-jul-2007 23:40:34 org.hibernate.connection.DriverManagerConnectionProvider co
nfigure
INFO: connection properties: {user=root, password=****}
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: RDBMS: MySQL, version: 5.0.22-community-nt
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: JDBC driver: MySQL-AB JDBC Driver, version: mysql-connector-java-5.0.5 ( $
Date: 2007-03-01 00:01:06 +0100 (Thu, 01 Mar 2007) $, $Revision: 6329 $ )
25-jul-2007 23:40:34 org.hibernate.dialect.Dialect <init>
INFO: Using dialect: org.hibernate.dialect.MySQLDialect
25-jul-2007 23:40:34 org.hibernate.transaction.TransactionFactoryFactory buildTr
ansactionFactory
INFO: Transaction strategy: org.hibernate.transaction.JDBCTransactionFactory
25-jul-2007 23:40:34 org.hibernate.transaction.TransactionManagerLookupFactory g
```

```
etTransactionManagerLookup
INFO: No TransactionManagerLookup configured (in JTA environment, use of read-wr
ite or transactional second-level cache is not recommended)
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Automatic flush during beforeCompletion(): disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Automatic session close at end of transaction: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: JDBC batch size: 15
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: JDBC batch updates for versioned data: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Scrollable result sets: enabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: JDBC3 getGeneratedKeys(): enabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Connection release mode: auto
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Maximum outer join fetch depth: 2
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Default batch fetch size: 1
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Generate SQL with comments: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Order SQL updates by primary key: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory createQueryTranslatorFact
ory
INFO: Query translator: org.hibernate.hql.ast.ASTQueryTranslatorFactory
25-jul-2007 23:40:34 org.hibernate.hql.ast.ASTQueryTranslatorFactory <init>
INFO: Using ASTQueryTranslatorFactory
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Query language substitutions: {}
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: JPA-QL strict compliance: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Second-level cache: enabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Query cache: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory createCacheProvider
INFO: Cache provider: org.hibernate.cache.NoCacheProvider
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Optimize cache for minimal puts: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
```

```
INFO: Structured second-level cache entries: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Echoing all SQL to stdout
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Statistics: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Deleted entity synthetic identifier rollback: disabled
25-jul-2007 23:40:34 org.hibernate.cfg.SettingsFactory buildSettings
INFO: Default entity-mode: pojo
25-jul-2007 23:40:34 org.hibernate.impl.SessionFactoryImpl <init>
INFO: building session factory
25-jul-2007 23:40:35 org.hibernate.impl.SessionFactoryObjectFactory addInstance
INFO: Not binding factory to JNDI, no JNDI name configured
HibernateUtil - constructor end
listRegisters - Hibernate configured
Hibernate:
    select
        register0_.id as id0_,
        register0_.dateCreation as dateCrea2_0_,
        register0_.name as name0_
    from
        coregest.register register0_
Registros leidos: 0
listRegisters - Ends
testApp  fin guardado un registro
insertReegister - Beginning
insertRegister  getted session
insertRegister  guardado el registro Registro1
Hibernate:
    insert
    into
        coregest.register
        (dateCreation, name, id)
    values
        (?, ?, ?)
insertRegister cerrada la tx
insertRegister - End
testApp - End
testApp2 - Beginning
insertReegister - Beginning
insertRegister  getted session
insertRegister  guardado el registro Registro2
Hibernate:
```

```
    insert

    into

        coregest.register

        (dateCreation, name, id)

    values

        (?, ?, ?)

insertRegister cerrada la tx

insertRegister - End

listRegisters - Begins

listRegisters - Hibernate configured

Hibernate:

    select

        register0_.id as id0_,

        register0_.dateCreation as dateCrea2_0_,

        register0_.name as name0_

    from

        coregest.register register0_

Registros leidos: 2

net.sf.coregest.core.model.Register@13c7378

net.sf.coregest.core.model.Register@1c0f2e5

listRegisters - Ends

testApp2 - End

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.094 sec


Results :


Tests run: 2, Failures: 0, Errors: 0, Skipped: 0


[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 5 seconds

[INFO] Finished at: Wed Jul 25 23:40:35 CEST 2007

[INFO] Final Memory: 4M/7M

[INFO] ------------------------------------------------------------------------


D:\workspace\WorkspaceCoregest\coregest-core>
```

Podemos ver varias cosas en el listado:

- Al principio está la etapa de compilación, que variará según vayamos ejecutando los test.
- Luego veremos que al ejecutar el test, en la primera llamada a HibernateUtil se configuara Hibernate.
- Luego veremos cómo se van haciendo las consultas SQL
- Finalmente veremos que los test han tenido éxito.

A medida que vayamos avanzando en la aplicación podremos ir activando y desactivando los test, para que sólo se hagan test exhaustivos en las compilaciones críticas.

Mi recomendación es que creemos módulos separados para cada capa e intentemos que los cambios en un módulo no afecten mucho a los demás, de manera que se hagan cada vez menos necesarios los test unitarios. Por ejemplo, si conseguimos un modelo de datos estable, veremos reducida la necesidad de ejecutar estas pruebas, pero si constantemente estamos modificando el modelo, vamos a necesitar mucho tiempos para las pruebas unitarias.

Prueba de la aplicación desde un tomcat externo

Para probar la aplicación, incluyendo su war, puede ser conveniente preparar una instalación de tomcat. Para ello basta seguir estos pasos:

1. Descargamos el tomcat en formato zip, por ejemplo en mi caso la 5.5.17 y lo descomprimimos en una carpeta, en mi caso D:\apps\apache-tomcat-5.5.17
2. Creamos en nuestro espacio de trabajo la carpeta catalina_base y copiamos en ella las carpetas server, temp, logs y webapps del tomcat
3. En nuestro espacio de trabajo creamos el siguiene fichero de texto

D:\workspace\WorkspaceCoregest\startup coregest.bat

```
REM script de arranque del tomcat para coregest



set CATALINA_BASE D:/workspace/WorkspaceCoregest/catalina_base



d:

cd D:\apps\apache-tomcat-5.5.17\BIN\

STARTUP.BAT
```

4. En nuestro espacio de trabajo nos creamos un enlace a http://localhost:8080/coregest/index.jsp para poder abrir el navegador rápidamente.
5. copiamos el coregest.war de la carpeta coregest\target a catalina_base\webapps

Ahora ya podemos arrancar el tomcat rápidamente, aprovechando el tomcat instalado en d:\apps y sin modificarlo. Así podremos probar muchas aplicaciones diferentes con una misma instalación del tomcat.

Conclusión

En este tutorial hemos aprendido a integrar Hibernate en nuestro proyecto de aplicación Web, de manera que se integra tanto con Maven como con Eclipse. También hemos escrito el código básico de uso de Hibernate por nuestra aplicación, así como la configuración básica del mismo. Todo esto lo hemos probado mediante pruebas unitarias que se ejecutan automáticamente durante la compilación del módulo de la lógica de gestión. Todo esto no es más que el esqueleto de una aplicación, ahora nos queda mejorarlo:

- Hayu que completar el modelo de datos de la aplicación con sus clases java y sus tablas en la base de datos
- Hay que mejorar el código de soporte de Hibernate en la aplicación (soporte de multihilos, JNDI, etc)
- Hay que escribir el código de los gestores de las entidades de nuestra aplicación. Cada gestor (manager) es responsable de obtener objetos de una entidad
- También podemos integrar el EntityManager de Hibernate en la aplicación. Las HibernateTools nos crean unos gestores básicos que utilizan el EntityManager.

Si quereis avanzar más, podeis seguir nuestros tutoriales, que esperamos os sean de ayuda.

Enlaces de interés

http://www.manning.com/bauer2/ Aquí está el capítulo 2 del libro "Java Persistence with Hibernate"

www.hibernate.org, aquí están los tutoriales y manual de referencia de Hibernate

http://www.natidea.com/wiki/index.php?title=Hibernate_JPA Este artículo viene con ejemplos de integración de Hibernate

SOME RIGHTS RESERVED This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 2.5 License.
Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Autentia S.L. Somos expertos en:
J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..
y muchas otras cosas

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
Probando entornos para JSF	En este tutorial os mostramos con ejemplos como utliizar dos conocidos entornos de desarrollo para JSF: Exadel Studio y Sun Studio Creator
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: aplicación multimódulo	En este artículo se va a abordar el desarrollo de una aplicación Myfaces JSF con Maven que sea multimódulo.
Hibernate 3.1, Colecciones, Fetch y Lazy	En este tutorial vamos a ver cómo se comportan ciertas relaciones, y cómo podemos optimizar las consultas a la base de datos con Hibernate
Pruebas unitarias Web para aplicaciones JSF	En este tutorial se puede encontrar una introducción y un análisis de los diferentes frameworks disponibles para realizar pruebas unitarias web de aplicaciones JSF
Proyecto con JSF Myfaces, Maven y Eclipse	En este tutorial vamos a aprender a construir una aplicación básica JSF (Java Server Pages) utilizando el Maven 2.0 y las bibliotecas de MyFaces. Lo mejor de todo es que para crear el ejemplo no vamos a programar ni una línea.
JSF y comparativa con Struts	Os mostramos los pasos necesarios para empezar a utilizar JSF (Java Server Faces) y su comparación / relación con Struts
Hibernate Tools y la generación de código	En este tutorial vamos a ver como usar estas herramientas para hacer el esqueleto de una pequeña aplicación, de manera muy sencilla, generando código a partir de las tablas creadas en la base de datos.
Como hacer un componente de JSF	En este tutorial Alejandro Pérez nos enseñará como construir nuestro propio componente en JSF mediante un ejemplo
Hibernate y las anotaciones de EJB 3.0	En este tutorial Alejandro Pérez nos muestra las ventajas que nos aporta Hibernate y las anotaciones de EJB 3.0

Introducción a Hibernate	Cesar Crespo nos enseña como utilizar unos de los sistemas más extendidos de mapeo de objetos a estructuras relacionales (tablas de base de datos)

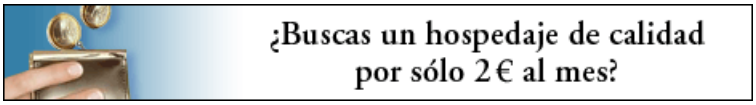
Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

Patrocinados por enredados.com .... Hosting en Castellano con soporte Java/J2EE



www.AdictosAlTrabajo.com Opimizado 800X600