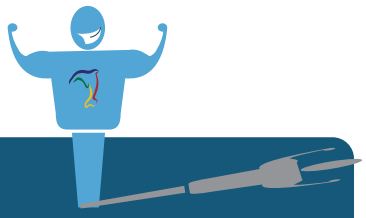
autentia

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...

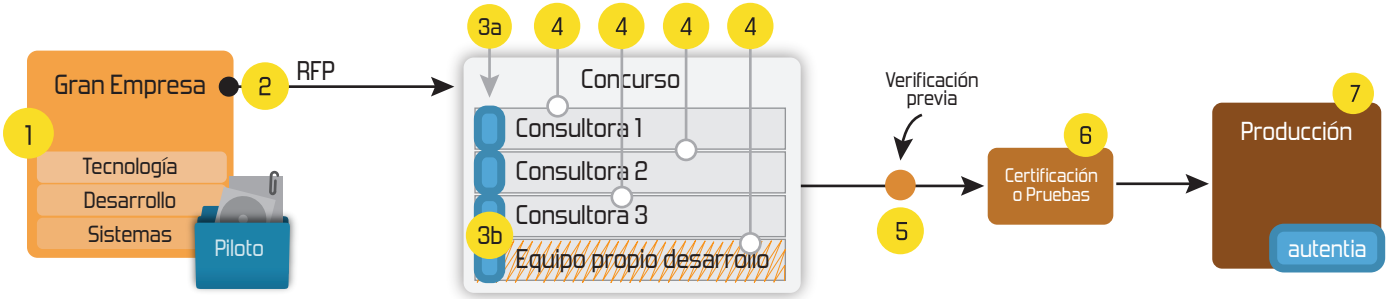
1. Desarrollo de componentes y proyectos a medida



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

NUEVO ¿Quieres saber cuánto ganas en relación al mercado? pincha aquí...

Ver cursos que ofrece Autentia

Descargar comics en PDF y alta resolución

[¡NUEVO!] 2008-12-01

2008-11-17

2008-09-01

2008-07-31

Estamos escribiendo un libro sobre la profesión informática y estas viñetas formarán parte de él. Puedes opinar en la seccion comic.

Tutorial desarrollado por

Enrique Viñé Lerma

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Informática por la Universidad Politécnica de Madrid.

Puedes encontrarme en Autentia

Somos expertos en Java/J2EE

Catálogo de servicios de Autentia

Descargar (6,2 MB)

Descargar en versión comic (17 MB)

AdictosAlTrabajo.com es el Web de difusión de conocimiento de Autentia.

autentia real business solutions

Catálogo de cursos

Descargar este documento en formato PDF: eventosenhibernateii.pdf

Fecha de creación del tutorial: 2009-01-28

Eventos en Hibernate (Parte II)

Índice de contenidos

1. Introducción

Este tutorial es continuación del tutorial "Eventos en Hibernate (Parte I)". Aquí vamos a centrarnos en el uso de los eventos definidos en la especificación EJB3 gracias a Hibernate Entity Manager. Podemos asociar métodos de retrollamada a diferentes eventos de forma que se ejecuten de manera automática. Por ejemplo, esto es muy útil para calcular el valor de campos de nuestros objetos que no se persisten en la base de datos, pero se pueden calcular en función de otros; podemos asignar un método de retrollamada para que se ejecute al cargar la entidad y calcule el valor de dicho campo.

En este tutorial vamos a continuar con el ejemplo que vimos anteriormente, en el cual teníamos una entidad usuarios con un campo edad no persistido, que se calculará automáticamente a partir de la fecha de nacimiento.

2. Entorno

Para la realización de este tutorial se han utilizado las siguientes herramientas:

- Hardware: Portátil Asus G50Vseries (Core Duo P8600 2.4GHz, 4GB RAM, 320 GB HD).
- Sistema operativo: Windows Vista Ultimate.
- Eclipse Ganymede 3.4.1, con el plugin Q (http://code.google.com/p/q4e) para Maven
- JDK 1.6.0
- Maven 2.0.9
- MySql 5.1.30
- Hibernate 3

3. Reconfigurando la aplicación de ejemplo

Últimos tutoriales

Últimas ofertas de empleo

Para poder utilizar un EntityManager debemos añadir la siguiente dependencia al fichero pom.xml.

```
view plain print ?
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>3.4.0.GA</version>
</dependency>
```

4. Un DAO nuevo

El Dao que habíamos utilizado en el ejemplo anterior no nos vale ahora, ya que utilizaba una Session de Hibernate para manejar nuestras entidades. Si queremos tener toda la potencia de las anotaciones EJB3 debemos utilizar un EntityManager, que es equivalente a Session pero amplía sus capacidades permitiéndonos utilizar toda la semántica de EJB3.

Es importante realizar las operaciones de persistencia dentro de una transacción para poder hacer COMMIT. De lo contrario por defecto se hará ROLLBACK y nuestras acciones no tendrán el efecto esperado. Nosotros mismos deberemos controlar en nuestro código la creación del EntityManager y el inicio y fin de las transacciones, ya que nuestra aplicación va a ejecutarse fuera de un contenedor J2EE.

```
view plain print ?
package modelo;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;


public class UsuarioEntityManagerDao {

    public void guardarUsuario(Usuario usuario) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( "manager1");
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = null;

        try {
            tx = em.getTransaction();
            tx.begin();
            em.persist(usuario);
            tx.commit();
        } finally {
            em.close();
            emf.close();
        }
    }

    public List<Usuario> buscarUsuarios() {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( "manager1");
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = null;
        List<Usuario> usuarios = new ArrayList<Usuario>();

        try {
            tx = em.getTransaction();
            tx.begin();
            usuarios = em.createQuery( "from Usuario").getResultList();
            tx.commit();
        } finally {
            em.close();
            emf.close();
        }

        return usuarios;
    }
}
```

5. Configuración de Hibernate

Para crear el EntityManager, hemos utilizado la clase Persistence, pasándole al método "createEntityManagerFactory" un nombre como parámetro. Este nombre identificará una unidad de persistencia en la que definiremos la configuración que será utilizada para crear el EntityManager. Este fichero se llamará "persistence.xml" y deberá ponerse dentro de la carpeta "META-INF", porque es allí donde lo buscará la EntityManagerFactory. El atributo name de la unidad de persistencia debe coincidir con el que le pasamos a la EntityManagerFactory.

El tipo de transacción se ha puesto como "RESOURCE_LOCAL". Esta es la única posibilidad que tenemos para un simple proyecto Java, pero si dispusiéramos de un contenedor J2EE podríamos utilizar "JTA", lo cual nos evitaría tener que iniciar por nosotros mismos las transacciones.

```
view plain print ?
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
    <persistence-unit name="manager1" transaction-type="RESOURCE_LOCAL">
        <class>modelo.Usuario</class>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLInnoDBDialect" />
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
            <property name="hibernate.connection.username" value="root" />
            <property name="hibernate.connection.password" value="root" />
            <property name="hibernate.connection.url"
                value="jdbc:mysql://localhost:3306/eventos?autoReconnect=true" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.show_sql" value="true" />
            <!--
                Alternativamente podemos utilizar un fichero hibernate.cfg.xml
                corriente
            -->
            <!--
                <property name="hibernate.ejb.cfgfile" value="hibernate.cfg.xml"/>
            -->
        </properties>
    </persistence-unit>
</persistence>
```

Una alternativa a poner la configuración de Hibernate en el fichero "persistence.xml" consiste en indicar un fichero de configuración de hibernate de donde

cogerla (la línea que aparece comentada en el código anterior). Podemos incluso mezclar ambas configuraciones, pero en caso de conflicto siempre prevalecerá la que pongamos en el fichero "persistence.xml".

6. Probar de nuevo la aplicación

Ahora sólamente nos queda probar la aplicación que hemos rehecho, para ver si funciona como es debido. Para ello modificamos nuestras dos clases de prueba para que utilicen el nuevo Dao.

Aquí está el código modificado de nuestra clase PruebaInsercion.

```
view plain  print  ?
01. package prueba;
02.
03. import java.util.Calendar;
04. import java.util.Date;
05. import java.util.GregorianCalendar;
06.
07. import modelo.Usuario;
08. import modelo.UsuarioEntityManagerDao;
09.
10. public class PruebaInsercion {
11.     public static void main(String[] args) {
12.         Calendar cal = new GregorianCalendar();
13.         cal.set(1970, 10, 5);
14.         nuevoUsuario("Juan", "Lopez", cal.getTime());
15.         cal.set(1976, 3, 16);
16.         nuevoUsuario("Enrique", "Perez", cal.getTime());
17.         cal.set(1973, 5, 9);
18.         nuevoUsuario("Laura", "Iglesias", cal.getTime());
19.     }
20.
21.     private static void nuevoUsuario(String nombre, String apellidos, Date edad) {
22.         UsuarioEntityManagerDao dao = new UsuarioEntityManagerDao();
23.         Usuario usuario = new Usuario(nombre, apellidos);
24.         usuario.setNacimiento(edad);
25.         dao.guardarUsuario(usuario);
26.     }
27. }
```

Y aquí tenemos la clase PruebaListado, que utiliza también el nuevo Dao.

```
package prueba;

import java.util.List;

import modelo.Usuario;
import modelo.UsuarioEntityManagerDao;

public class PruebaListado {
        public static void main(String[] args) {
                UsuarioEntityManagerDao dao = new UsuarioEntityManagerDao();
                List<Usuario> usuarios = dao.buscarUsuarios();
                for (Usuario usuario : usuarios) {
                        System.out.printf("%s %s tiene %s años\n", usuario.getNombre(),
                                        usuario.getApellidos(), usuario.getEdad());
                }
        }
}
```

Si hemos seguido el tutorial anterior tendremos que borrar la tabla de nuestro esquema de base de datos, para comprobar que vuelve a crearse y a cargarse con datos correctos.

Ejecutamos primero la inserción y comprobamos que se han añadido correctamente los usuarios a nuestra base de datos:

```
Query Area
1 SELECT * FROM usuario u;
```

id	apellidos	nacimiento	nombre
1	Lopez	1970-11-05 01:09:21	Juan
2	Perez	1976-04-16 01:09:21	Enrique
3	Iglesias	1973-06-09 01:09:21	Laura

Ahora ejecutamos la prueba de listado y observamos... ¡Vaya, esto me suena! Lo hemos vuelto a hacer, ¿no teníamos que capturar el evento?.

```
Juan Lopez tiene 0 años
Enrique Perez tiene 0 años
Laura Iglesias tiene 0 años
```

7. Configurar eventos

Este es el apartado que más me gusta del tutorial, porque sólamente tenemos que escribir una palabra: @PostLoad. Esta anotación delante de nuestra función "calcularEdad" es todo lo que tenemos que hacer para que nuestra función sea llamada después de cargar los datos de la entidad. Deberemos también añadir el import necesario.

```
view plain print ?
...

import javax.persistence.PostLoad;

...

@PostLoad
public void calcularEdad() {
Calendar cumple = new GregorianCalendar(); Calendar ahora = new
GregorianCalendar();

...
```

Si volvemos a ejecutar la función que lista los usuarios, obtendremos el resultado esperado:

```
Hibernate: select usuario0_.id a
Ejecutando nuestro oyente
Ejecutando nuestro oyente
Ejecutando nuestro oyente
Juan Lopez tiene 38 años
Enrique Perez tiene 32 años
Laura Iglesias tiene 35 años
```

A continuación se muestran todas las anotaciones que pueden utilizarse para que los métodos de la entidad sean llamados de forma automática al producirse un evento en su ciclo de vida. Los métodos anotados de esta manera se llaman "Callback Methods" o métodos de retrollamada.

Podemos asignar varias anotaciones para un mismo método, y designar varios métodos de retrollamada en una misma entidad. Pero no podremos asignar dos métodos diferentes para el mismo evento.

Tipo	Descripción	Interfaz del oyente
@PrePersist	Ejecutado antes de realizar la persistencia (INSERT) del objeto.	org.hibernate.event.PreInsertEventListener
@PreRemove	Ejecutado antes de realizar el borrado (DELETE) del objeto.	org.hibernate.event.PreDeleteEventListener
@PostPersist	Ejecutado después de realizar la persistencia del objeto.	org.hibernate.event.PostInsertEventListener
@PostRemove	Ejecutado después de realizar el borrado del objeto.	org.hibernate.event.PostDeleteEventListener
@PreUpdate	Ejecutado antes de la actualización (UPDATE) del objeto.	org.hibernate.event.PreUpdateEventListener
@PostUpdate	Ejecutado después de la actualización del objeto.	org.hibernate.event.PostUpdateEventListener
@PostLoad	Ejecutado después de la carga o refresco del objeto.	org.hibernate.event.PostLoadEventListener

8. Añadir nuestros propios oyentes

La anotaciones anteriores asignan los oyentes predeterminados de Hibernate a los métodos de retrollamada. También podemos asignar nuestros propios oyentes anotando la clase con @EntityListeners. Podremos pasarle como parámetro nuestra clase oyente o un array de clases oyentes:

`@EntityListeners(value=Oyente.class) ó @EntityListeners(value={Oyente1.class, Oyente2.class, ...})`

Las clases oyentes asignadas de esta manera deberán implementar métodos para cada evento que quieran manejar, los cuales se marcarán con la anotación correspondiente. Estos métodos recibirán como parámetro una instancia de la entidad sobre la que ocurre el evento.

Vamos a implementar un oyente que rejuvenezca a nuestros usuarios, pero esta vez cambiando su fecha de nacimiento.

```
view plain print ?
package oyentes;

import java.util.Calendar;
import java.util.GregorianCalendar;

import javax.persistence.PostLoad;

import modelo.Usuario;

public class EJB3LiftingUsuarioOyente {

    @PostLoad
    public void postLoad(Usuario usuario) {
        System.out.println("Quitando 3 años a " + usuario.getNombre());
        Calendar cal = new GregorianCalendar();
        cal.setTime(usuario.getNacimiento());
        cal.add(Calendar.YEAR, 3);
        usuario.setNacimiento(cal.getTime());
    }
}
```

Para que se ejecute el oyente, tendremos que utilizar la anotación @EntityListeners en la entidad Usuario:

```
view plain print ?
...

@Entity
@EntityListeners(value={EJB3LiftingUsuarioOyente.class})
public class Usuario {

...
```

Si ejecutamos el listado, veremos la siguiente salida:

```
Hibernate: select usuario0_.id as id0_, usuario0_.apellidos as apellidos0_, u
Quitando 3 años a Juan
Quitando 3 años a Enrique
Quitando 3 años a Laura
Hibernate: update Usuario set apellidos=?, nacimiento=?, nombre=? where id=?
Hibernate: update Usuario set apellidos=?, nacimiento=?, nombre=? where id=?
Hibernate: update Usuario set apellidos=?, nacimiento=?, nombre=? where id=?
Juan Lopez tiene 35 años
Enrique Perez tiene 29 años
Laura Iglesias tiene 32 años
```

Si nos fijamos en las sentencias update de Hibernate veremos que esto ha tenido un efecto secundario. Hemos cambiado la fecha de nacimiento de los usuarios en la base de datos. Recordemos que según la semántica de EJB3, si tenemos un objeto manejado por el gestor de persistencia no es necesario

hacer persist, merge, update o nada parecido para guardar los cambios que hagamos al mismo. Nosotros modificamos el valor de los campos del objeto y el gestor de persistencia se encargará automáticamente de reflejar esos cambios en la base de datos. Así que... este nuevo lifting parece más duradero que el anterior ;)

9. Diferentes formas de gestionar los eventos

Tenemos diferentes maneras de conseguir que un método se ejecute al producirse un evento para una entidad. En caso de existir varios métodos a ejecutar, estos se ejecutarán en el orden siguiente:

1. A través de la anotación @EntityListeners en la entidad o una superclase de la misma. Estos oyentes se ejecutarán en el orden en que se añadan al array "value". Es posible evitar que se ejecuten los oyentes definidos por la anotación @EntityListeners de las superclases utilizando la etiqueta @ExcludeSuperclassListeners en una entidad.
2. Oyentes de las superclases de la entidad definidos en el fichero de configuración (primero la superclase de nivel superior).
3. Oyentes de la propia entidad definidos en el fichero de configuración.
4. Métodos de retrollamada (anotaciones @PostLoad...) para las superclases de la entidad (primero la de nivel superior).
5. Métodos de retrollamada para la entidad.

Ya hemos visto cómo utilizar los métodos de retrollamada y también la anotación @EntityListeners. También podemos asignar los EntityListeners y los métodos de retrollamada por medio de un fichero de configuración xml. Deberemos indicar en nuestro "persistence.xml" el fichero de mapeo a utilizar:

view plain print ?

```
...

    <persistence-unit name="manager1" transaction-type="RESOURCE_LOCAL">
        <mapping-file>orm.xml</mapping-file>
        <class>modelo.Usuario</class>

...
```

A través del fichero de mapeo podemos asignar oyentes por defecto para todas las entidades y oyentes particulares para cada entidad e, incluso, definir los métodos de retrollamada de nuestra entidad sin utilizar las anotaciones. Por ejemplo, en el siguiente fichero xml hemos definido para la entidad Usuario el oyente EJB3LiftingUsuarioOyente y asociado el evento post-load al método calcularEdad.

view plain print ?

```
<?xml version="1.0" encoding="UTF-8"?>

<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm orm_1_0.xsd"
                 version="1.0"
        >
    <persistence-unit-metadata>
        <persistence-unit-defaults>
            <entity-listeners></entity-listeners>
        </persistence-unit-defaults>
    </persistence-unit-metadata>

    <entity class="modelo.Usuario">
        <entity-listeners>
            <entity-listener class="oyentes.EJB3LiftingUsuarioOyente">
                <post-load method-name="postLoad"/>
            </entity-listener>
        </entity-listeners>

        <post-load method-name="calcularEdad"/>
    </entity>
</entity-mappings>
```

Es importante destacar que no pueden definirse dos métodos de retrollamada diferentes para el mismo evento, por lo que deberemos eliminar las anotaciones @PostLoad de nuestra entidad Usuario y de nuestra clase EJB3LiftingUsuarioOyente.

10. Conclusiones

A la vista de los resultados de este tutorial, podemos extraer las siguientes conclusiones:

- Un EntityManager nos permite utilizar Hibernate con la semántica definida por JPA para la especificación de EJB3.
- Podemos mantener nuestra antigua configuración de Hibernate añadiendo una referencia a la misma en nuestro persistence.xml. A pesar de eso, podremos sobreescribir ciertas propiedades en el nuevo fichero de configuración.
- Con el uso de las anotaciones de EJB3 es muy sencillo definir métodos de retrollamada para los eventos del ciclo de vida de nuestras entidades, y también es más sencillo crear nuestras propias clases oyente.

Y eso es todo por ahora. En el próximo tutorial de esta serie veremos una forma de utilizar las anotaciones de EJB3 para definir eventos de retrollamada utilizando sólamente la clase Session de Hibernate, es decir, sin utilizar un EntityManager.

Una vez más, desde Autentia esperamos que este tutorial os sea de ayuda si os encontráis en la necesidad de ejecutar eventos en Hibernate.

Recuerda

Servicio de notificaciones:

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales.

Formulario de subcripción a novedades:

E-mail [ ] Aceptar

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	pdf
Hibernate Validator, y como definir las validaciones sobre los objetos de negocio	Hibernate Validator permite introducir valiaciones en los objetos de negocio (POJOs) en invocarlas según intesese. Álex nos explica cómo, en este tutorial	2008-06-14	2032	-	pdf
Creación de una aplicación con Spring e Hibernate desde 0	Este tutorial vamos a explicar paso a paso cómo crear una pequeña aplicación usando Spring e Hibernate con anotaciones partiendo desde 0	2008-02-15	8979	-	pdf
Como implementar el método equals(Object) en objetos persistentes de Hibernate, y otras consideraciones.	En este tutorial Alejandro Pérez nos muestra cómo implementar el método equals(Object) en objetos persistentes de Hibernate, y otras consideraciones.	2008-02-05	2408	-	pdf
Ficheros de mapeo de Hibernate desde las clases	Vamos a ver cómo las HibernateTools nos permiten generar de manera automática los ficheros de mapeo de Hibernate a partir de anotaciones en las clases	2008-06-02	3325	-	pdf
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	2007-06-25	8791	-	pdf
Comparativa entre Hibernate y EJB3 en la Capa de Persistencia	El presente documento pretende dar algunas luces a la comparativa entre la opción de usar Hibernate y/ó EJB3 para la capa de persistencia	2007-08-16	8581	-	pdf
Uso de Filtros en Hibernate	En este tutorial Carlos García nos muestra un ejemplo del uso de Filtros en Hibernate.	2008-08-14	2211	-	pdf
Spring + Hibernate + Anotaciones = Desarrollo Rápido en Java	Alejandro Pérez nos enseña lo fácil y rápido que es desarrollar en Java usando Spring e Hibernate, y usando anotaciones	2008-05-14	9117	-	pdf
Eventos en Hibernate (parte I)	Este es el primer tutorial de una serie de tutoriales que tienen por objetivo mostrar el uso de eventos en Hibernate. En este primer tutorial se muestra cómo utilizar oyentes con una SessionFactory.	2009-01-27	78	-	pdf
Hibernate y el mapeo de la herencia	En este tercer tutorial de la saga vamos a ver en este tutorial como implementar las relaciones de herencia con las anotaciones de JPA	2007-06-27	6237	-	pdf

Nota: