

# EJB 3.X

---

VICTOR CUSTODIO

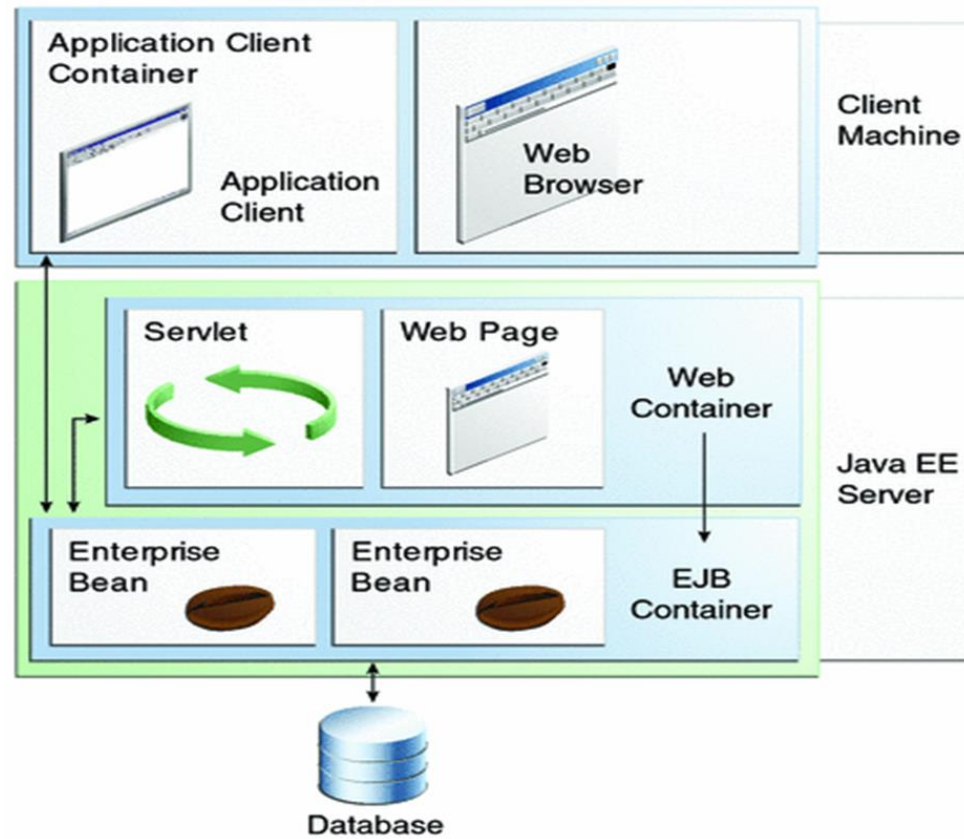
# ¿Que es EJB?

---

Enterprices Java Beans (EJB): una de las Api que forman parte del estándar de construcción de aplicaciones empresariales JEE.

- Su especificación detalla como los servidores de aplicaciones proveen objetos de negocio desde el lado del servidor.
- Es un modelo de programación que nos permite construir aplicaciones Java mediante objetos ligeros (como POJO's).
- Nos permite centrarnos en el código de la lógica de negocio del problema que deseamos solucionar y deja el resto de responsabilidades(Seguridad, transaccionalidad, concurrencia, etc) al contenedor de aplicaciones donde se ejecutará la aplicación.
- Esta basado en la Arquitectura RMI enfocada a problemas empresariales y la Arquitectura JEE

# El contenedor de aplicaciones



# El contenedor EJB

---

Es un entorno (una aplicación) que provee los servicios comunes a nuestra aplicación, gestionando por nosotros. Dichos servicios incluyen

- creación/mantenimiento/destrucción de nuestros objetos de negocio,
- servicios mencionados en el punto anterior(concurrencia, transaccionalidad...) entre otros.
- Aunque el contenedor es responsable de la gestión y uso de dichos recursos/servicios, podemos interactuar con él para que nuestra aplicación haga uso de los servicios que se ofrecen

# Los EJB

---

Una clase no actuará como un componente EJB hasta que haya sido empaquetado, desplegado en un contenedor EJB y accedido por dicho contenedor (por ejemplo a petición de un usuario).

Una vez que una clase definida como EJB haya sido desplegada en el contenedor, se convertirá en uno de los tres siguientes componentes

- Session Bean (clases de lógica de negocio)
- Message-Driven Bean (Actúan de forma asíncrona para trabajar con mensajes)
- Entity Bean (Representaciones de información- JPA)

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

## Message-Driven Beans: Solo conceptos Básicos

## Entity Beans.

# Session Beans

---

Son los componentes que contienen la lógica de negocio que requieren los clientes de nuestra aplicación.

Son accedidos a través de un stub(también llamado *vista*)

Tras realizar una solicitud al contenedor el cliente obtiene una vista del Session Bean, pero no el Session Bean real. Esto permite al contenedor realizar ciertas operaciones sobre el Session Bean real de forma transparente para el cliente (como gestionar su ciclo de vida, solicitar una instancia a otro contenedor trabajando en paralelo, etc).

# Session Beans

---

Los componentes Session Bean pueden ser de tres tipos:

- Stateless Session Beans (No guardan estado)
- Stateful Session Beans (Si guardan estado para cada cliente)
- Singletons (Una única instancia para todo el contenedor)



# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

Entity Beans.

# Session Beans – Ejemplo Stateless (SLSB)

---

Primer EJB, la interfaz:

```
@Remote
public interface MiInterfaceEJB {
    public String saluda(String nombre);
}
```

• Primer EJB, la implementación:

```
import javax.ejb.Remote;
import javax.ejb.Stateless;

@Stateless
public class PrimerEJB implements MiInterfaceEJB {
    public String saluda(String nombre) {
        if(nombre == null) {
            return "Hola desconocido";
        }

        return "Hola " + nombre;
    }
}
```

# Session Beans – Ejemplo Stateless (SLSB)

---

`@Stateless` define nuestra clase como un Session Bean de tipo Stateless y una vez desplegado en un contenedor EJB, este lo reconocerá como un componente EJB SLSB que podremos usar.

`@Remote` permite a nuestro EJB ser invocado remotamente (esto es, desde fuera del contenedor EJB). Si omitimos esta anotación, el EJB sería considerado como *Local* y solo podría ser invocado por otros componentes ejecutándose dentro del mismo contenedor

# Session Beans – Ejemplo Stateless (SLSB)

---

## Implementación.

- Creamos un proyecto EJB en Eclipse
- Creamos un nuevo EJB y su interfaz remota con el código de ejemplo
- Ejecutamos el servidor de aplicaciones . Debe ser un servidor con contenedor EJB(En nuestro caso Wildfly).

# Session Beans – Ejemplo Stateless (SLSB)

---

Tenemos un Servidor de aplicaciones con un contenedor EJB o (Módulo) que ofrece la interfaz `MiInterfazEJB`, ahora podremos usar los EJB de varias formas:

- Desde un cliente java externo (Conseguimos el contexto del servidor de aplicaciones y hacemos un lookup (similar a RMI)
- Desde un modulo Web en el mismo servidor de aplicaciones (por inyección)

Siempre usando la interfaz remota.

# Session Beans – Ejemplo Stateless (SLSB)

---

Si queremos usar el EJB desde otro EJB o desde cualquier otra clase del mismo contenedor EJB, necesitamos declarar una interfaz Local y que nuestro EJB la implemente:

```
@Local
public interface MiInterfaceEJBLocal {
    public String saluda(String nombre);
}
```

```
import javax.ejb.Remote;
import javax.ejb.Stateless;

@Stateless
public class PrimerEJB implements MiInterfaceEJB,
MiInterfazEJBLocal {
    public String saluda(String nombre) {
        if(nombre == null) {
            return "Hola desconocido";
        }

        return "Hola " + nombre;
    }
}
```

# Session Beans – Ejemplo Stateless (SLSB)

---

Ahora también podremos usar nuestro EJB

- Desde otros EJB's en el mismo contenedor (por inyección)
- Desde cualquier clase java en el contenedor EJB (por inyección)

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

Entity Beans.



# Inyección de Dependencias

---

¿Qué es la inyección?

- Es un proceso por el cual el contenedor puede *inyectar* en un componente recursos que son necesarios.  
Ejemplo:

@Stateless

```
public class UnComponente {  
    private OtroComponente dependencia;  
  
    public String metodo() {  
        return dependencia.otroMetodo();  
    }  
}
```

- Un Componente EJB necesita usar otro componente EJB.

# Inyección de Dependencias

---

@Local

```
@Stateless(name="otroComponente")
public class OtroComponente {
    public String otroMetodo() {
        // ...
    }
}
```

@Stateless

```
public class UnComponente {
    @EJB(beanName="otroComponente")
    private OtroComponente dependencia;

    public String metodo() {
        return dependencia.otroMetodo();
    }
}
```

- Gracias a la anotación @EJB el contenedor sabe que componente instanciar, inicializar y por último inyectar en la variable dependencia.
- En el ejemplo, se registra un Session Bean con nombre otroComponente (@Stateless(name="otroComponente")), el cual puede ser accedido mediante inyección de dependencia gracias a @EJB(beanName="otroComponente")

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

Entity Beans.

# EJB context y Session context

---

A veces, necesitamos acceder al contexto de ejecución del componente que se está usando:

- **EJBContext** (Contexto de EJB) es una interface que provee acceso al contexto de ejecución asociado a cada instancia de un componente EJB. A través de él podemos acceder, por ejemplo, al servicio de seguridad, transacciones u obtener el Timer Service (para tareas programadas)
- **SessionContext** (Contexto de Sesión) es una interface que implementa EJBContext .Añade métodos que permiten el uso de servicios adicionales. A través de él podemos, por ejemplo, obtener una referencia al Session Bean actual para poder pasarla a otro Session Bean como argumento de un método; esto es necesario, pues el contenedor no permite pasar el Session Bean actual usando una referencia de tipo `this`

# EJB context y Session context

---

Podemos acceder al contexto de sesión asociado a la instancia del componente actual mediante inyección:

@Local

@Stateless

```
public class MiBean {
```

```
    @Resource
```

```
    private SessionContext contexto;
```

```
    // ...
```

```
}
```

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- **Stateless Session Beans**
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

Entity Beans.

# Stateless Session Beans (SLSB)

---

Los SLSB son componentes que no requieren mantener un estado entre diferentes invocaciones.

Un cliente debe asumir que diferentes solicitudes al contenedor de un mismo SLSB pueden devolver vistas a objetos diferentes.

Un SLSB puede ser compartido (y probablemente lo será) entre varios clientes.

Por todo esto, los SLSB son creados y destruidos a discrección del contenedor, y puesto que no mantienen estado son muy eficientes a nivel de uso de memoria y recursos en el servidor.

# Stateless Session Beans(SLSB)

---

Ciclo de vida. Solo dos estados:

- No existe (Does not exists)
- Preparado en pool (Method-ready pool)

El primer estado, no existe: la instancia del SLSB no ha sido creada aún

El segundo estado, *Preparado en pool*:

- Representa una instancia del SLSB que ha sido instanciada y construida por el contenedor, se encuentra en el pool lista para recibir invocaciones por parte de un cliente. Cuando esta invocación sucede, la instancia es extraída del pool y asociada a una referencia que es pasada al cliente.
- Durante el arranque del contenedor el pool es poblado, cuando no existan suficientes instancias en el pool, el contenedor creará más



# Stateless Session Beans(SLSB)

---

Durante la transición entre el primer estado y el segundo, el contenedor realizará tres operaciones en el siguiente orden:

- Instanciación del SLSB
- Inyección de cualquier recurso necesario y de dependencias
- Ejecución de un método dentro del SLSB marcado con la anotación `@PostConstruct`, si existe

Cuando el contenedor no necesita una instancia de SLSB se realiza una transición en sentido inverso: de preparado en pool al estado no existe. Durante esta transición se ejecutará, si existe, un método anotado con `@PreDestroy` (Pre destrucción)

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

Entity Beans.

# StateFull Session Beans(SFSB)

---

- Mantienen estado entre distintas invocaciones de un mismo cliente
- Cada SFSB está dedicado de manera exclusiva a un único cliente durante todo su ciclo de vida
- No son almacenados en un pool, pues no son reusados
- Un SFSB almacena información a consecuencia de las operaciones que realiza en él su cliente asociado, y dicha información (estado) debe estar disponible en invocaciones posteriores
- A pesar de mantener estado, un SFSB no es persistente, de manera que dicho estado se pierde cuando la sesión del cliente asociado termina

# StateFull Session Beans(SFSB)

---

Ciclo de vida, 3 estados:

- No existe (Does not exists)
- Preparado (Method-ready)
- Pasivo (Passive)

El segundo estado, *preparado*, representa una instancia del SFSB que ha sido construida e inicializada, y está lista para servir llamadas de su cliente asociado

El tercer estado, *pasivo*, representa un SFSB que, después de un periodo de inactividad, es persistido temporalmente para liberar recursos del servidor.

# StateFull Session Beans(SFSB)

---

Durante la transición entre el primer estado y el segundo, el contenedor realizará tres operaciones en el siguiente orden:

- Instanciación del SLSB
- Inyección de cualquier recurso necesario y de dependencias
- Ejecución de un método dentro del SLSB marcado con la anotación `@PostConstruct`, si existe

Un SFSB en estado *preparado* puede realizar una transición a cualquiera de los otros dos estados de su ciclo de vida: *no existe* o *pasivo*.

- La transición al estado *no existe* ocurre cuando:
  - El cliente ejecuta un método dentro del SFSB anotado con `@Remove`
  - El SFSB sobrepasa un periodo de inactividad establecido (timeout)
- La transición al estado pasivo se produce cuando el contenedor decide liberar recursos, persistiendo de manera temporal el estado del SFSB

# StateFull Session Beans(SFSB)

---

## Pasivación y Activación.

- *Pasivación.* El contenedor puede decidir liberar recursos persistiendo de manera temporal el estado de dicho SFSB, liberando de esta manera memoria del sistema (u otros recursos). Antes de pasivar el contenedor ejecutará el método anotado con `@PrePassivate`
- *Activación.* Si durante la pasivación el cliente realiza una llamada al SFSB, el contenedor recreará en memoria la instancia persistida y procederá con dicha llamada. Después de activar un SFSB el contenedor ejecutará el método anotado con `@PostActivate`

# Ejemplo SFSB

---

@Stateful

```
public class Carrito implements Serializable {  
    private Map articulosEnCarrito = new HashMap();  
    private BaseDeDatos bbdd;  
  
    public void añadirArticulo(Articulo articulo, int cantidad) {  
        // añadir la cantidad de cierto artículo al carrito  
    }  
  
    public void eliminarArticulo(Articulo articulo, int cantidad) {  
        // eliminar la cantidad de cierto artículo del carrito  
    }  
  
    public void vaciarCarrito() {  
        // vaciar el carrito  
    }  
}
```

# Ejemplo SFSB

---

@Remove

```
public void finalizarCompra() {  
    // procesar el pedido  
}
```

@PostConstruct

@PostActivate

```
private void inicializar() {  
    // obtener conexión con la base de datos  
}
```

@PrePassivate

@PreDestroy

```
private void detener() {  
    // liberar conexión con la base de datos  
}
```



# Ejemplo SFSB

---

## **@Remove**

```
public void finalizarCompra() {  
    // procesar el pedido  
}
```

## **@PostConstruct**

### **@PostActivate**

```
private void inicializar() {  
    // obtener conexión con la base de datos  
}
```

## **@PrePassivate**

### **@PreDestroy**

```
private void detener() {  
    // liberar conexión con la base de datos  
}
```

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

Entity Beans.

# Singleton Session Beans

---

Es un nuevo tipo de Session Bean introducido en la especificación EJB 3.1. Este componente se basa en el patrón de diseño del mismo nombre

Este patrón de diseño garantiza que de una clase dada solamente pueda crearse una instancia, con un punto de acceso global para acceder a dicha instancia.

Conlleva un alto rendimiento dentro del contenedor para este tipo de Session Bean.

# Singleton Session Beans - Concurrency

---

A diferencia de los SLSB y SFSB un Singleton, multiples llamadas en paralelo pueden estar produciendose en un momento dado a su única instancia,

El componente debe garantizar que un hilo de ejecución no está interfiriendo con otro hilo de ejecución, produciendo resultados incorrectos

La especificación EJB 3.1 nos permite controlar la concurrencia de un Singleton de dos formas:

- Concurrencia Gestionada por el Contenedor (CMC, Contained-Managed Concurrency)
- Concurrencia Gestionada por el Bean (BMC, Bean-Managed Concurrency)

# Singleton Session Beans - Concurrency

---

- Concurrency Gestionada por el Contenedor (CMC, Contained-Managed Concurrency).
- Es ofrecida por defecto (si no se indica nada)

```
@Singleton
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
class MiSingleton {
    // ...
} @Lock(LockType.READ) //indicamos Read para que no sea una función bloqueante,(solo se lee)
public String metodo() {
    // ...
}
```

# Singleton Session Beans - Concurrency

---

- Concurrency Gestionada por el Bean(BMC, Bean-Managed Concurrency).

@Singleton

@ConcurrencyManagement(ConcurrencyManagementType.BEAN)

class OtraClaseConcurrente {

    // Gestión explícita de la concurrencia

}

- Debemos ocupar a base de synchronized , wait, notify....

# Singleton Session Beans

---

- Ciclo de vida:

- No Existe (Does not exists)
- Preparado (Method-ready)

Solamente pasa de un estado a otro una vez. Tenemos opciones para elegir cuando crear la instancia:

- `@Startup` :Indicamos al contenedor que cree la instancia al arracar.
- `@DependsOn(OtroSingleton)`: Requerimos la instancia de un Singleton antes de instanciar la clase donde se defina la etiqueta.

- Al igual que con los componentes SLSB y SFSB, disponemos de las anotaciones `@PostConstruct` y `@PreDestroy`

# Singleton Session Beans - Ejemplo

@Singleton

public class SistemaDeLog {

private FileWriter writer;

---

private enum Nivel {

DEBUG, INFO, ERROR

}

@PostConstruct

protected void inicializar() throws IOException {

writer = new FileWriter("aplicacion.log", true);

}

@PreDestroy

protected void detener() throws IOException {

writer.flush();

writer.close();

}

@Lock(LockType.WRITE)

public void debug(String mensaje) {

escribirMensajeEnArchivo(Nivel.DEBUG, mensaje);

}

@Lock(LockType.WRITE)

public void info(String mensaje) {

escribirMensajeEnArchivo(Nivel.INFO, mensaje);

}



# Singleton Session Beans - Ejemplo

```
@Lock(LockType.WRITE)
```

```
public void error(String mensaje) {
```

```
    escribirMensajeEnArchivo(Nivel.ERROR, mensaje);
```

---

```
}
```

```
private void escribirMensajeEnArchivo(Nivel nivel, String mensaje) {
```

```
    String cabecera = generarCabecera(nivel);
```

```
    try {
```

```
        writer.write(cabecera + mensaje + "\n");
```

```
    } catch (IOException ioe) {
```

```
        throw new RuntimeException(ioe);
```

```
    }
```

```
}
```

```
private String generarCabecera(Nivel nivel) {
```

```
    String fechaMasHoraActual = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date());
```

```
    StringBuilder cabecera = new StringBuilder();
```

```
    cabecera.append("[");
```

```
    cabecera.append(nivel.name());
```

```
    cabecera.append("] ");
```

```
    cabecera.append(fechaMasHoraActual);
```

```
    cabecera.append(" - ");
```

```
    return cabecera.toString();
```

```
}
```

# A continuación veremos

---

Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

**Message-Driven Beans: Solo conceptos Básicos**

Entity Beans.

# Message-Driven Beans: conceptos básicos

---

Los componentes de tipo Message-Driven Bean (MDB - Bean Dirigido por Mensajes) son componentes asíncronos de tipo *listener*.

Un MDB no es más que un componente que *espera* a que se le envíe un mensaje, y realiza cierta acción cuando finalmente recibe dicho mensaje

Algunas propiedades de los componentes MDB son:

- No mantienen estado
- Son gestionados por el contenedor (transacciones, seguridad, concurrencia, etc)

# A continuación veremos

---

## Session Beans.

- Ejemplo
- Inyección
- EJBContext
- Stateless Session Beans
- Statefull Sessions Beans
- Singleton

Message-Driven Beans: Solo conceptos Básicos

## **Entity Beans.**

# Entidades

---

JavaBeans, clases simples java con atributos y métodos de acceso (getters y setters) y constructor por defecto vacío.

Las entidades, a diferencia del resto de componentes EJB, son objetos Java reales que son manejados entre componentes (o entre un cliente y un componente) en su forma original, nunca a través de proxys/vistas. Podemos crearlos con sentencias new.

Pertenecen a la especificación JPA (que ya hemos estudiado). A partir de JavaEE 5 EJB absorbió JPA.

# Entidades

---

## Ciclo de vida

- Gestionada (Attached)
- No gestionada (Detached)

En el primer estado, la entidad se encuentra *gestionada* por el servicio de persistencia: cualquier cambio que realicemos en su estado se verá reflejado en la base de datos subyacente.

En el segundo estado, la entidad es un objeto Java regular, y cualquier cambio que realicemos en su estado no será sincronizado con la base de datos subyacente. La entidad puede ser, por ejemplo, enviada a través de una red (mediante serialización).

# Entidades – Unidad de Persistencia

---

Una *unidad de persistencia* (persistence unit) representa un conjunto de entidades que pueden ser mapeadas a una base de datos, así como la información necesaria para que la aplicación JPA pueda acceder a dicha base de datos.

Se define mediante un archivo llamado *persistence.xml*

# Entidades – Unidad de Persistencia

---

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

```
<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_2_0.xsd"
version="2.0">
```

```
  <persistence-unit name="introduccionEJB">
```

```
    <!-- configuración de acceso a la base de datos -->
```

```
    <!-- lista de entidades que pueden ser mapeadas -->
```

```
  </persistence-unit>
```

```
</persistence>
```



# Entidades – Unidad de Persistencia

---

Podemos definir más de una unidad de persistencia por aplicación, declarando cada una de ellas mediante el elemento XML <persistence-unit>. Cada unidad de persistencia debe seguir estas dos reglas:

- Debe proporcionar un nombre (identidad) a través del cual pueda ser llamado
- Debe conectar a una sola fuente de datos (data source)

# Entidades – Contexto de Persistencia

---

Un contexto de persistencia representa un conjunto de instancias de entidades que se encuentran gestionadas en un momento dado. Existen dos tipos de contextos de persistencia:

- Limitados a una transacción (Transaction-scoped)
- Extendidos (Extended)

Dentro de un contexto de persistencia *limitado a una transacción*, todas las entidades gestionadas pasarán a estar no gestionadas cuando dicha transacción finalice. Los cambios realizados después de finalizar la transacción no serán sincronizados con la base de datos.

Dentro de un contexto de persistencia *extendido*, todos los cambios que realicemos en el estado de las entidades gestionadas por el contexto de persistencia se sincronizarán con la base de datos en el momento en que entremos en una nueva transacción

# Entidades - EntityManager

---

Mediante la interface EntityManager (Gestor de entidades) tenemos acceso al servicio de persistencia de nuestro contenedor.

A diferencia que con sólo JPA con EJB3.1 podemos obtener una instancia de EntityManager en nuestros componentes EJB mediante inyección de dependencias:

```
@Stateless
```

```
public class MiSlsb {
```

```
    @PersistenceContext(unitName="introduccionEJB")
```

```
    private EntityManager em;
```

```
    // operaciones del SLSB
```

```
}
```

# Entidades –configuración

---

Como quien manejará ahora la conexión con la base de datos es el contenedor, debemos configurar Jpa con Jboss, para ello añadimos el datasource al standalone.xml de Jboss:

```
<datasource jndi-name="java:jboss/datasources/mysql" pool-name="mySqlDS" enabled="true" use-java-context="true">

    <connection-url>jdbc:mysql://localhost:3306/prueba</connection-url>

    <driver>mysql</driver>

    <security>

        <user-name>root</user-name>

        <password>root</password>

    </security>

</datasource>

<drivers>

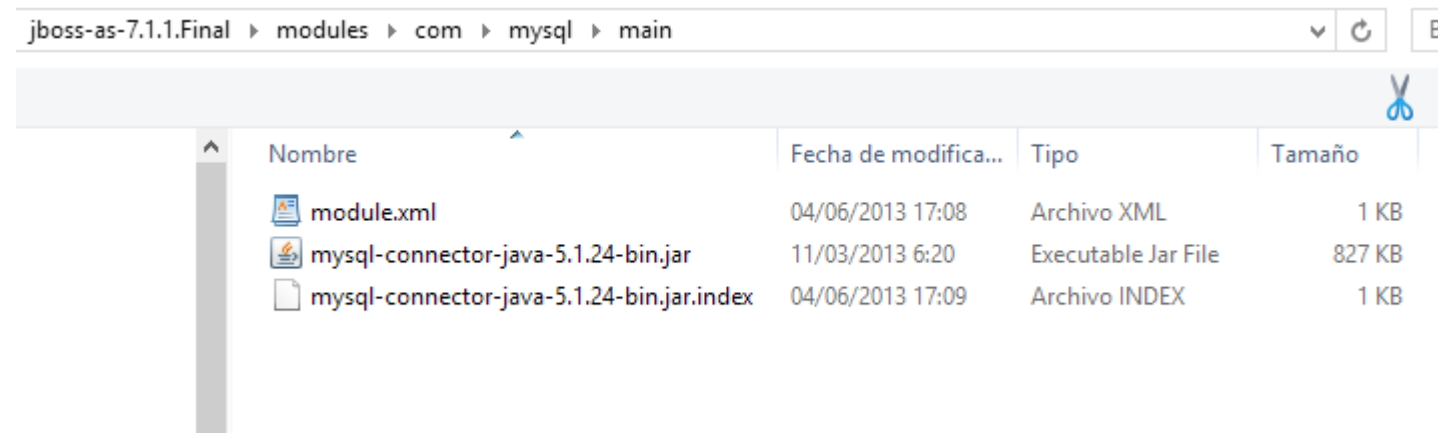
    <driver name="mysql" module="com.mysql"/>

</drivers>
```




# Entidades –configuración

---

Y añadimos el driver-conector a los módulos de Jboss:



The screenshot shows a file explorer window with the path `jboss-as-7.1.1.Final > modules > com > mysql > main`. The table below lists the files in this directory.

Nombre	Fecha de modifica...	Tipo	Tamaño
 module.xml	04/06/2013 17:08	Archivo XML	1 KB
 mysql-connector-java-5.1.24-bin.jar	11/03/2013 6:20	Executable Jar File	827 KB
 mysql-connector-java-5.1.24-bin.jar.index	04/06/2013 17:09	Archivo INDEX	1 KB

# Entidades –configuración

---

Añadimos propiedades JPA a nuestro proyecto EJB en eclipse.(Hibernate- JPA 2.1)

- Proyecto Ejb -> configuración -> convert in JPA Project

Modificamos el persistence.xml para indicar que la fuente de datos estará gestionada por el contenedor y que será la siguiente, con esta entrada:

- `<jta-data-source>java:jboss/datasources/mysql</jta-data-source>`

# Entidades –configuración

---

Recordar añadir al persistence.xml la siguiente etiqueta si queremos que nos cree la base de datos:

```
<property name="hibernate.hbm2ddl.auto"  
value="create-drop"/>
```

# Entidades –configuración

---

Ya estamos listos para crear la primera aplicación completa(de base de datos a web con EJB en la arquitectura JEE)