



Videotraining >

Lección 2. Enterprise JavaBeans

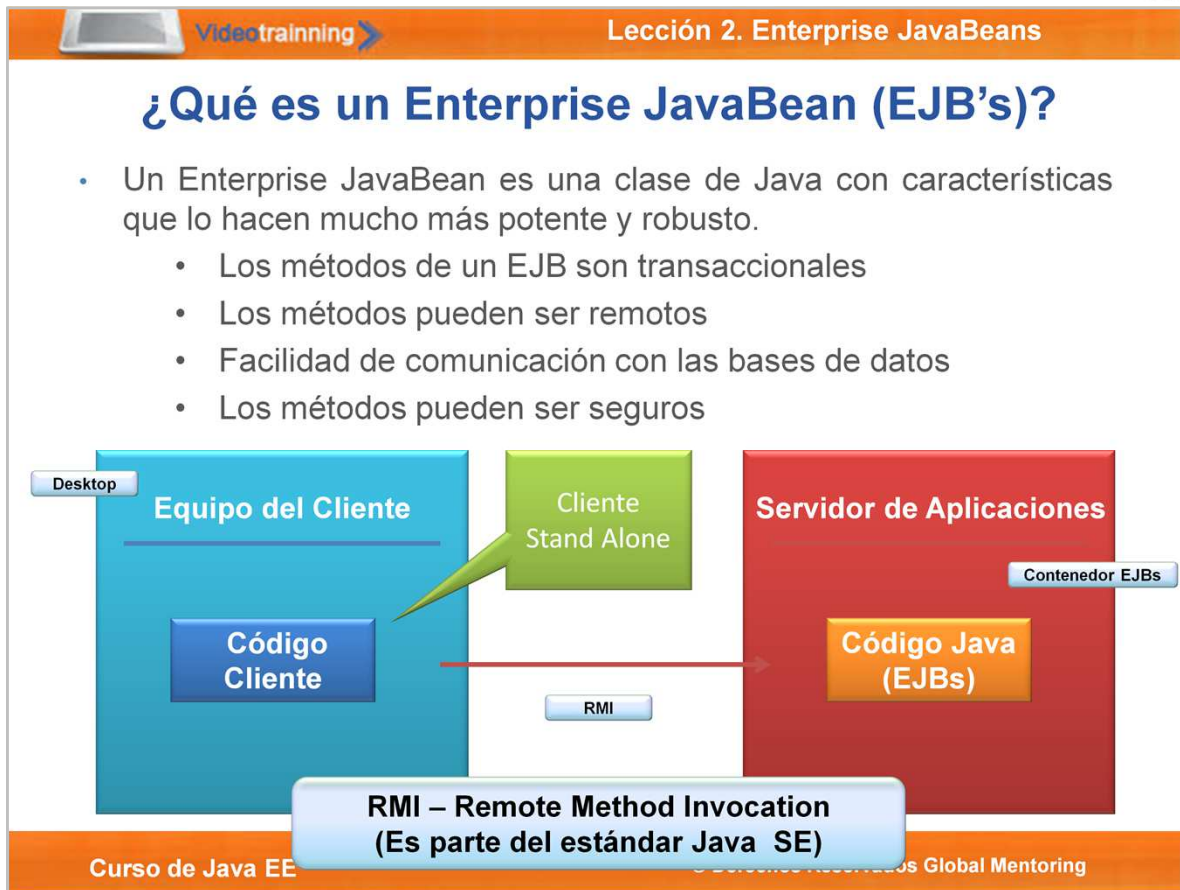


Lección 2

Enterprise JavaBeans
(EJB's)

www.globalmentoring.com.mx

© Derechos Reservados Global Mentoring



Los Enterprise Java Beans (EJB) es código Java del lado del Servidor. Normalmente tienen la lógica de negocio de nuestra aplicación, y por lo tanto cubren el rol de la capa de servicio de nuestras aplicaciones Java, según se estudió en la lección anterior.

Al día de hoy los EJB's son clases puras de Java (POJO's) los cuales al ser desplegados en un Servidor de Aplicaciones permiten reducir la complejidad de programación, agregando robustez, reusabilidad y escalabilidad a nuestras aplicaciones empresariales de misión crítica.

Hoy más que nunca la versión EJB 3.1 pueden ser programados una vez y ejecutados en cualquier servidor de aplicaciones Java que soporte el estándar Java EE 6. Los EJB's ya han cumplido más de una década desde su aparición, y al día de hoy son una tecnología muy probada y que brinda beneficios tales como seguridad, transaccionalidad, multi-threading, entre muchas características más, todo esto a través del servidor de aplicaciones Java.

A diferencia de un JavaBean, que es una clase pura de Java, un Enterprise JavaBean es una clase de Java con características que lo hacen mucho más potente y robusto:

- Los métodos de un EJB son transaccionales.
- Los métodos pueden ser remotos.
- Facilidad de comunicación con las bases de datos.
- Los métodos pueden ser seguros.
- Los métodos pueden ser asíncronos.
- Entre muchas características más.

En la figura podemos observar que el código Java del lado del Servidor es un EJB, el cual puede ser ejecutado por una aplicación, conocida como Cliente. Este cliente realiza una petición al componente EJB, pudiendo ser una llamada local (si se encuentra en el mismo servidor) o una llamada remota (si se encuentra fuera del servidor de aplicaciones). Si la llamada es remota, se utiliza el protocolo RMI (Remote Method Invocation), el cual es parte de la versión estándar de Java.



En una arquitectura típica Java EE, los EJB juegan el rol de la capa de Servicio, donde es común encontrar muchas de las reglas de negocio de nuestra aplicación.

Una regla de negocio son las normas o políticas de la empresa u organización, por ejemplo, si un cliente ha sido leal a un producto por cierto número de años, se le puede aplicar un descuento extra por determinado monto de compra. Este tipo de decisiones se aplican automáticamente por medio de los sistemas, y la capa de negocio es la encargada de ejecutar estas reglas.

Los EJB's al ejecutarse dentro de un contenedor EJB y a su vez dentro de un servidor de aplicaciones Java, tiene a su disposición varias características que puede utilizar, tales como:

- Seguridad por medio
- Llamadas Asíncronas
- Llamadas Remotas por medio de RMI
- Manejo de Transacciones por medio de JTA
- Exposición de reglas de negocio por medio de Servicios Web (JAX-WS o JAX-RS)
- Servicio de Inyección de Dependencias por medio de CDI
- Servicio de Pool de Conexiones
- Manejo de Concurrencia Seguro (Tread-Safety)
- Manejo de Tareas Programadas (Scheduling)
- Manejo de Mensajería por medio de JMS
- Interceptors, permiten interceptar llamadas a métodos y agregar funcionalidad extra o complementaria por medio de AOP (Aspect Oriented Programming)

Los servidores de aplicaciones Java, también agregan otras características tales como: clustering, balance de cargas y tolerancia a fallos. Esto permite crear aplicaciones de misión crítica con operaciones 7/24 los 365 días del año. Así que independientemente del tipo de servidor de aplicaciones que utilicemos, tendremos todas estas características disponibles al crear y desplegar nuestros EJBs.

Contamos con un curso de administración del servidor JBoss, el cual pueden revisar en este link, en el cual se estudian configuración como Clustering, tolerancia a fallos y configuraciones de alta disponibilidad. Pueden revisar toda la información de este curso en el siguiente link:

<http://globalmentoring.com.mx/curso-jboss/>

Videotrainning >
Lección 2. Enterprise JavaBeans

Configuración y Tipos de EJB

Configuración de un Enterprise JavaBeans (EJB):

POJO



+

Anotación



=

EJB 3



Tipos de Enterprise JavaBeans:

- Stateless:** No guardan estado y se utiliza la anotación @Stateless
- Stateful:** Guardan estado y se utiliza la anotación @Stateful
- Singleton:** Solo existe una instancia en memoria y se utiliza la anotación @Singleton

Curso de Java EE
© Derechos Reservados Global Mentoring

En versiones previas a EJB 3.0, el programador debía crear varias clases e interfaces para hacer funcionar un EJB: una interface local o remota (o ambas), una interface de tipo **home** local o remota (o ambas), y un archivo de configuración xml, conocido como deployment descriptor.

Los EJB en su versión 3.0 promovió el uso de anotaciones para su configuración, y la versión 3.1 continúa agregando y simplificando la integración de tecnologías empresariales a través del concepto de anotaciones. Este concepto simplificó en gran medida el desarrollo de EJBs, y en general de toda la tecnología Java.

Existen diferentes tipos de beans, dependiendo de la función que se agrega a una arquitectura multicapas Java. Además esta organización permite entender mejor la configuración de una aplicación empresarial.

Debido a que las aplicaciones empresariales suelen ser complejas, se han definido los siguientes tipos de EJBs, según los requerimientos a cubrir.

EJB de Sesión: Un bean de sesión se invoca por el cliente para ejecutar una operación de negocio específica.

- Stateless:** Este tipo de EJB no mantiene ningún estado del usuario, es decir, no recuerda ningún tipo de información después de terminada una transacción.
- Stateful:** Este tipo de EJB, mantiene un estado de la actividad del cliente, por ejemplo, si se tiene un carrito de compras. Este estado se puede recordar incluso una vez terminada la transacción, pero si el servidor se reinicia esta información se pierde. El similar al alcance Session de una aplicación Web.
- Singleton:** Este tipo de beans utiliza el patrón de diseño *Singleton*, en el cual solamente existe una instancia en memoria de esta clase.

Otras clasificaciones que podemos encontrar son:

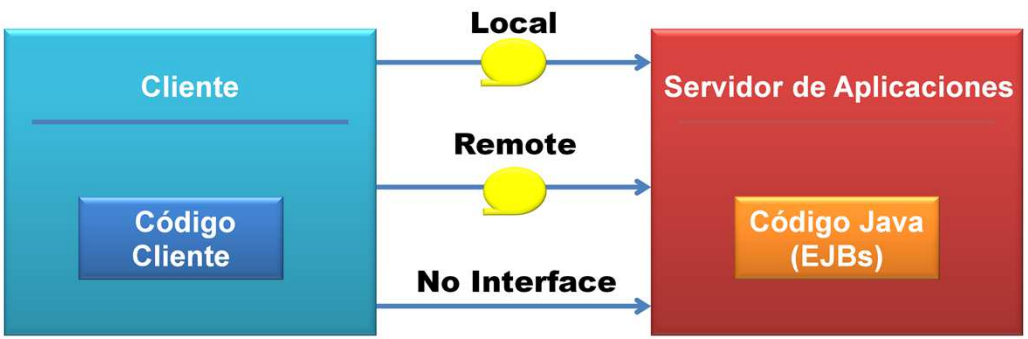
- EJB Timer:** Esta es una característica que se puede agregar a los beans, para que se ejecuten en un tiempo especificado (scheduling).
- Message-driven beans (MDBs):** Este tipo de beans se utiliza para enviar mensajes utilizando la tecnología JMS. El estudio de este tipo de beans queda fuera del alcance de este curso.
- Entity Beans:** Esta es una clasificación anterior a la versión 3.0 de los EJB, sin embargo al día de hoy el estándar JPA (Java Persistence API) ha sustituido a este tipo de beans. Así que, a menos que estemos utilizando una versión anterior a 3.0, se debería utilizar JPA en lugar de los Entity Beans.

 **Videotraining**

Lección 2. Enterprise JavaBeans

Formas de comunicarnos con un EJB

Existen diferentes formas de comunicarnos con un EJB:



- Interfaz Local: Se utiliza cuando el cliente se encuentra en el mismo servidor Java.
- Interfaz Remota: Se utiliza cuando el cliente se encuentra fuera del servidor Java
- No Interface: Es una variante y simplificación de los EJB locales.

Curso de Java EE

© Derechos Reservados Global Mentoring

La versión EJB 2.x incluía más conceptos y más complejidad de programación. En la versión 3 estos conceptos se han simplificado enormemente.

Los EJBs pueden ser configurados de la siguiente forma, con el objetivo de permitir la comunicación con sus métodos:

- **Interfaces de Negocio:** Estas interfaces contienen la declaración de los métodos de negocio que son visibles al cliente. Estas interfaces son implementadas por una clase Java.
- **Una clase Java (bean):** Esta clase implementa los métodos de negocio y puede implementar cero o más **Interfaces de Negocio**. Dependiendo del tipo de EJB, esta clase se debe anotar con `@Stateless`, `@Stateful` o `@Singleton` dependiendo del tipo de EJB que deseemos crear.

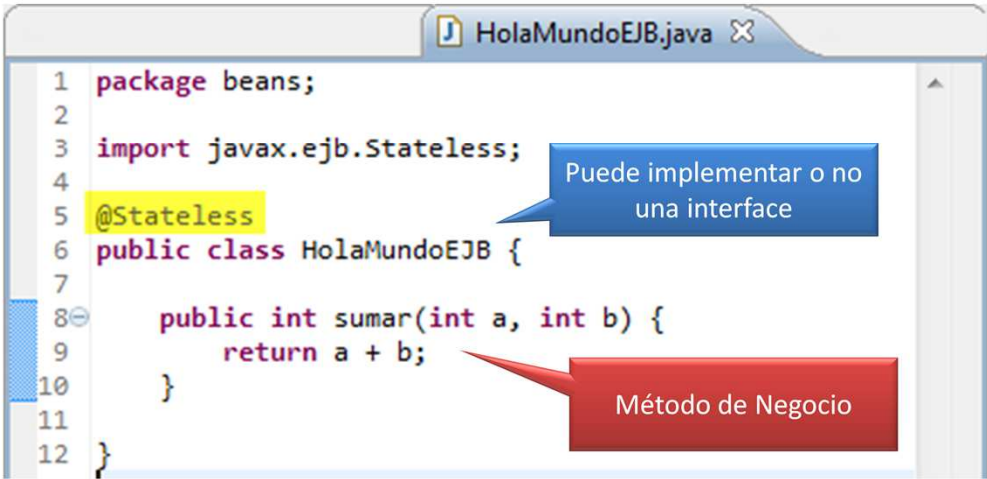
Como podemos observar en la figura, tenemos diferentes formas de comunicarnos con nuestro componente EJB.

- **Interfaz Local:** Se utiliza cuando el cliente se encuentra dentro del mismo servidor Java, de esta manera se evita la sobrecarga de procesamiento al utilizar llamadas remotas vía RMI.
- **Interfaz Remota:** Se utiliza cuando el código del cliente está fuera del servidor de aplicaciones Java (en una Java Virtual Machine distinta) y por lo tanto debemos hacer llamadas remotas para poder ejecutar los métodos del EJB.
- **No Interface:** Es una simplificación en la versión 3.1, ya que no se requiere de una interfaz para establecer la comunicación, siempre y cuando las llamadas sean locales, es decir, dentro del mismo servidor de aplicaciones Java.

Videotrainning ▶ Lección 2. Enterprise JavaBeans

Anatomía de un EJB

- En la siguiente figura podemos observar la estructura general de un EJB, el cual puede implementar o no una interface (local o remota), y puede tener uno o más métodos de negocio:



```
1 package beans;
2
3 import javax.ejb.Stateless;
4
5 @Stateless
6 public class HolaMundoEJB {
7
8     public int sumar(int a, int b) {
9         return a + b;
10    }
11
12 }
```

Curso de Java EE © Derechos Reservados Global Mentoring


Previo a la versión J2EE se requería crear varias clases para hacer funcionar a un EJB: una interfaz local o remota (o ambas), un interfaz home local o remota (o ambas) y un descriptor de despliegue xml. La versión Java EE 5 y EJB 3.0 simplificó dramáticamente esta configuración agregando el concepto de anotaciones, sin embargo todavía se requería agregar una interfaz a los EJB, local o remota.

Como se observa en la figura, la versión Java EE 6 y EJB 3.1 permite convertir una clase pura de Java (POJO: Plain Old Java Object) en un EJB simplemente agregando la anotación del bean correspondiente, por ejemplo `@Stateless`. Esto automáticamente hace que esta clase tenga características como métodos transaccionales, métodos con seguridad, y puede acceder al manejador de entidades (entity manager) y así persistir información en la base de datos, entre muchas características más. Todo esto simplemente agregando la anotación EJB.

Otra forma de configurar un EJB es utilizando el archivo descriptor `ejb-jar.xml`, el cual ya es opcional al día de hoy. Este archivo descriptor sobrescribe el comportamiento agregado con las anotaciones en las clases Java.

Aunque el código mostrado en la figura es muy simple, debemos hacer énfasis y recordar que un EJB es un componente que se ejecuta en un contenedor Java. Este ambiente de ejecución es el que permite agregar las características empresariales a nuestras clases Java permitiendo realizar llamadas remotas, inyección de dependencias, manejo de estados y ciclo de vida, pooling de objetos, manejo de mensajería, manejo de transacciones, seguridad, soporte de concurrencia, interceptores, manejo de métodos asíncronos, entre varias características más.

Todo esto ocurre simplemente haciendo deploy de esta clase Java al servidor de aplicaciones (sea embebido o no). Esto permite que el programador Java se enfoque en los métodos de negocio y delegue todas estas características de **requerimientos no funcionales** a los servidores de aplicaciones Java.

 Videotrainning
Lección 2. Enterprise JavaBeans

Cliente EJB vía JNDI

- JNDI es un API que permite encontrar servicios o recursos empresariales en un servidor de aplicaciones Java.
- Para encontrar un EJB a partir de la versión 3.1 podemos utilizar la siguiente sintaxis:

```
java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
```

- Por ejemplo:

```
public void iniciarContenedor() throws Exception {
    System.out.println("----Iniciando EJBContainer...");
    Map<String, Object> map = new HashMap<String, Object>();
    map.put(EJBContainer.APP_NAME, "miApp");
    ec = EJBContainer.createEJBContainer(map);
    ctx = ec.getContext();
    ejb = (HolaMundoEJB) ctx.lookup("java:global/miApp/classes/HolaMundoEJB!beans.HolaMundoEJB");
}
```

Nombre JNDI portable en Java EE 6

Curso de Java EE
© Derechos Reservados Global Mentoring

JNDI (Java Naming and Directory Interface) es un API que nos permite encontrar servicios o recursos en un servidor de aplicaciones Java.

En un inicio JNDI era la única manera de encontrar los componentes EJB, pero conforme se introdujo el concepto de EJB locales y el manejo de anotaciones existieron otras maneras de ubicar y proporcionar una referencia de los componentes empresariales que se necesitan, a este concepto se le conoce como inyección de dependencias.

Anterior a la versión JEE 6, no existía un nombre estándar para ubicar a los EJB por medio del API JNDI, por lo que cada servidor Java brindaba sintaxis distintas para ubicar a los componentes empresariales. Sin embargo, a partir de la versión Java EE 6, se introdujo un nombre global para ubicar a los componentes EJB.

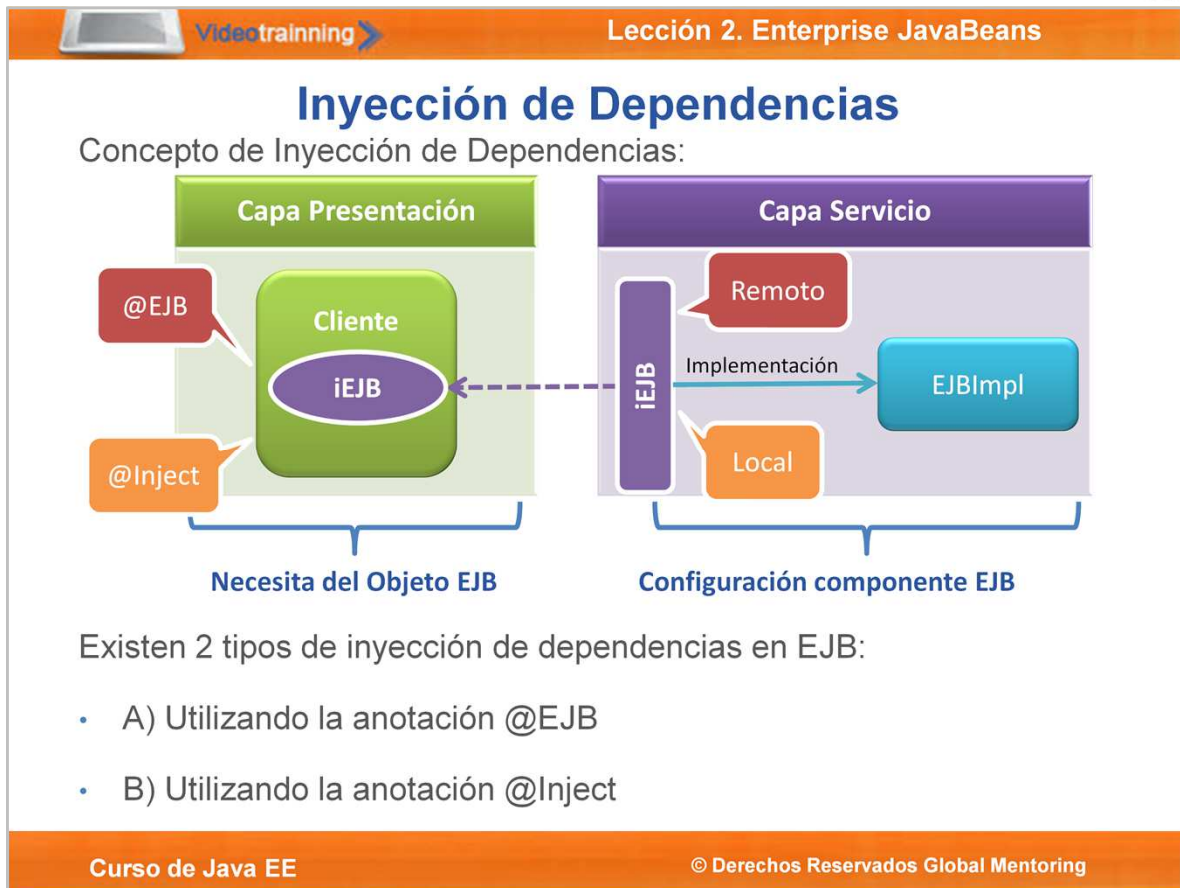
```
java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
```

Esto permite ubicar de manera estándar cualquier EJB en cualquier servidor de aplicaciones Java. El código básico para encontrar un EJB utilizando JNDI es:

```
HolaMundoEJB ejb = (HolaMundoEJB) contexto.lookup("java:global/classes/HolaMundoEJB");
```

O Incluyendo el nombre del paquete Java:

```
HolaMundoEJB ejb = (HolaMundoEJB)
contexto.lookup("java:global/classes/HolaMundoEJB!beans.HolaMundoEJB");
```



En la figura mostrada podemos observar un ejemplo en capas de una arquitectura empresarial. En este ejemplo la clase Cliente en la capa de presentación necesita del componente EJB de la capa de servicio, el cual puede estar ubicado en el mismo servidor (llamada local) o fuera del mismo (llamada remota). Para que la clase Cliente pueda utilizar el componente EJB, el servidor de aplicaciones puede proporcionar una referencia de dicho componente, a esto se le conoce como **Inyección de Dependencias**.

La inyección de dependencias revisa si existe en memoria un EJB ya sea con el mismo tipo o con el mismo nombre, según se especifique, y si existe ese objeto, el servidor de aplicaciones Java regresa una referencia para que pueda ser utilizado. En la versión empresarial Java EE 6 existen dos maneras de realizar la inyección de dependencias.

- a) Utilizando la anotación @EJB: Esta opción está disponible desde la versión Java EE 5, sin embargo es la forma de inyección de dependencias más básica. La anotación @EJB se recomienda cuando utilizamos llamadas remotas a los EJB, inyectar un recurso (JDBC DataSource, JPA, Web Service, etc) o si queremos mantener compatibilidad con Java EE 5. Ejemplo de código en el cliente:

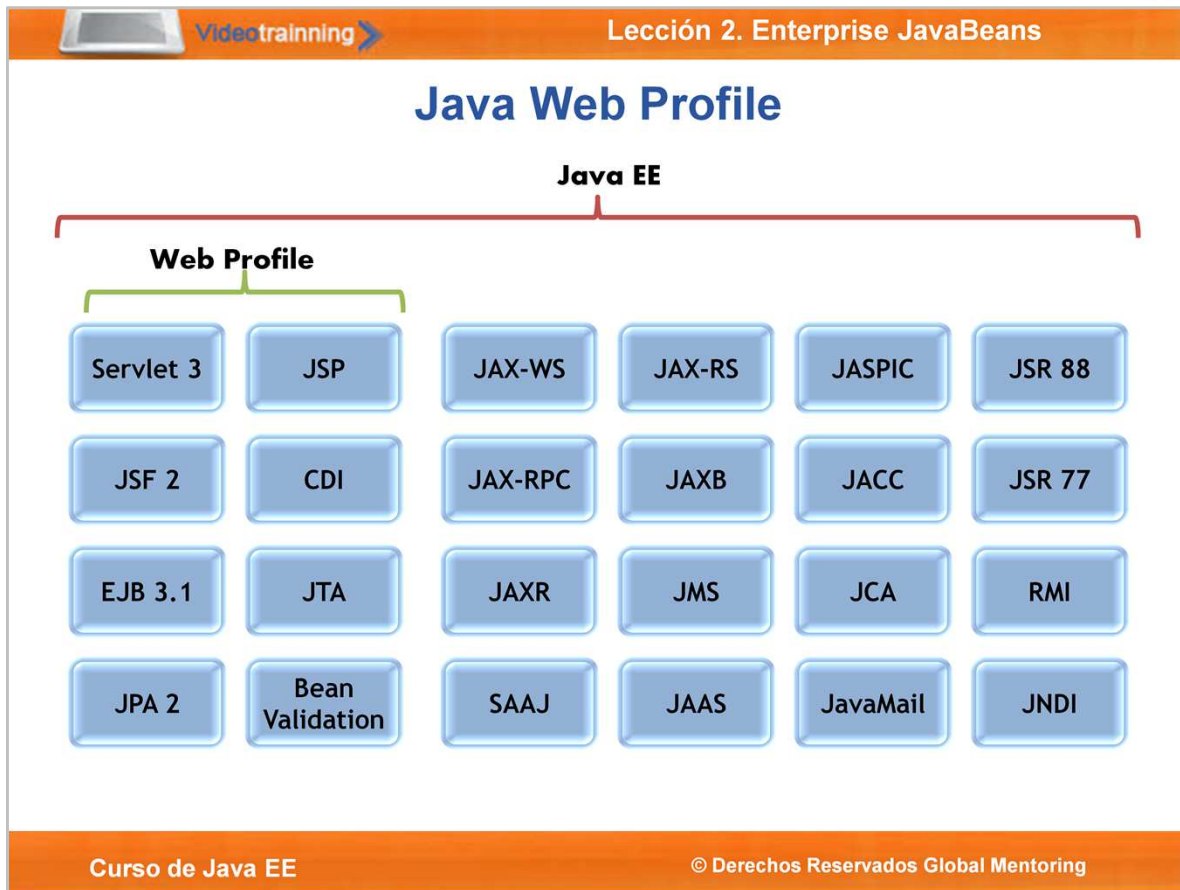
@EJB

```
private PersonaEJBRemote personaEJB;
```

- a) Utilizando la anotación @Inject: Esta forma de inyección de dependencia se apoya del concepto CDI (Context and Dependency Injection), y está disponible a partir de la versión Java EE 6. Esta forma es más flexible y robusta, ya que muchos de los conceptos fueron tomados de la experiencia de otros frameworks como Spring, los cuales tienen métodos de inyección de dependencias más poderosos y robustos. Para que el servidor de aplicaciones Java reconozca el concepto de CDI, se debe agregar un archivo llamado beans.xml. Se recomienda utilizar la anotación @Inject sobre @EJB en todos los casos, excepto cuando tenemos EJBs remotos o queremos mantener compatibilidad con Java EE 5. Ejemplo de código en el cliente.

@Inject

```
private PersonaEJB personaEJB;
```

En la figura podemos observar el API Java EE y en particular la relación con el perfil Web, el cual tiene acceso únicamente a ciertas APIs.

Esto surgió debido a que muchas de las aplicaciones Java EE no necesitaban de todo el poder ni las APIs tan robustas con las que cuenta, por lo tanto únicamente se agregaron a este perfil Web las APIs más comunes. La buena noticia es que podemos utilizar EJBs 3.1 en nuestras aplicaciones Web sin agregar la complejidad de configuración de los EJBs en versiones anteriores.

De hecho, en la versión Java EE 6 es posible utilizar EJBs locales sin necesidad de empaquetarlos por separado en un archivo .jar, sino únicamente utilizar un archivo .war. El tema de empaquetamiento lo revisaremos más adelante. Sin embargo, lo que debemos resaltar de esta figura es observar que tenemos acceso a los EJB, JPA, JTA, CDI, como las APIs más comunes que utilizaremos en nuestras aplicaciones empresariales.

Si necesitamos de otras APIs como Java Mail, Web Services, etc, será necesario utilizar un servidor de aplicaciones completo (full).

Seleccionar un tipo de perfil dependerá de los requerimientos actuales y futuros de nuestra aplicación empresarial, así que queda a consideración del Arquitecto/Programador la selección del perfil Java EE más adecuado a sus necesidades.

Lección 2. Enterprise JavaBeans		
Comparación de EJB y EJB Lite		
API Soportada	EJB Lite	Full EJB 3.1
Stateless beans	✓	✓
Stateful beans	✓	✓
Singleton beans	✓	✓
Message driven beans		✓
No Interfaces	✓	✓
Local Interfaces	✓	✓
Remote Interfaces		✓
Web service Interfaces		✓
Asynchronous Invocation		✓
Interceptors	✓	✓
Declarative security	✓	✓
Declarative transactions	✓	✓
Programmatic transactions	✓	✓
Timer Service		✓
EJB 2.x support		✓
CORBA Interoperability		✓

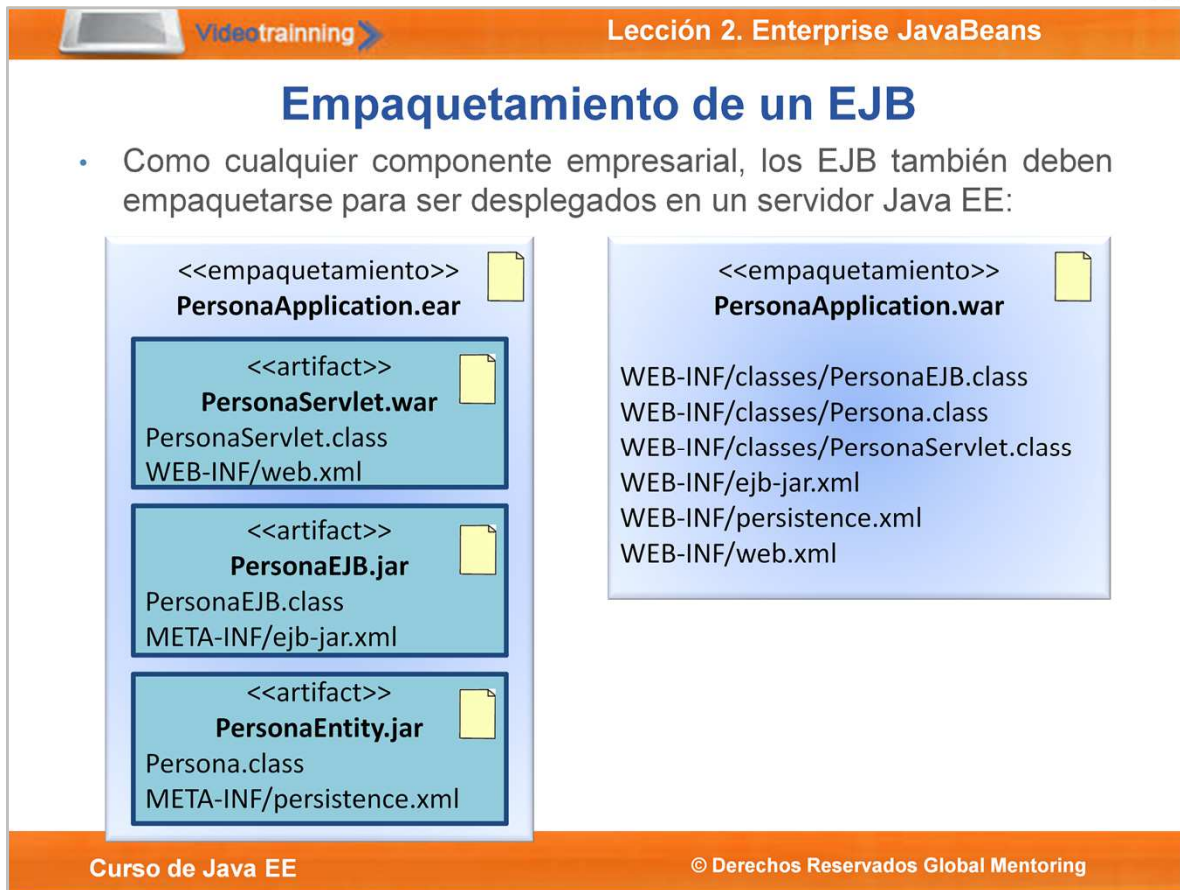
Los componentes predominantes en Java EE 6, sin duda son los EJBs, los cuales agregan de manera muy simple transaccionalidad, seguridad, entre más características que ya hemos comentado.

Según la lámina anterior, podemos utilizar el perfil Web de Java EE y utilizar EJBs. La especificación mínima de APIs que podemos utilizar en un perfil Web se conoce como EJB Lite. Las limitaciones de APIs que tenemos en el perfil Web son las limitantes que tenemos cuando utilizamos EJB, por ello el nombre de **lite**.

Podemos observar en la figura que si utilizamos el perfil Web y por consiguiente EJB Lite, únicamente tendremos acceso al API listada, excluyendo JMS, llamadas remotas a EJB, exposición de métodos del EJB como Web Service, llamadas Asíncronas, soporte para EJB 2.x, entre otras características que NO estarán disponibles.

Sin embargo podemos observar que muchos de los requerimientos empresariales más comunes SI tendremos acceso. Por ejemplo: Seguridad, Manejo de Transacciones, declaración de EJB Locales de tipo Stateless, Stateful, Singleton, con o sin Interface Local, etc.

Esto simplificó en gran medida las aplicaciones Web que necesitan de este tipo de requerimientos empresariales, sin sacrificar el performance ni el rendimiento de nuestra aplicación Java.



Debido a que una aplicación Java Empresarial incluye distintos tipos de componentes, tales como: Servlets, páginas JSF, Web Services, EJB, etc, estos componentes deben empaquetarse para ser desplegados en el servidor de aplicaciones Java.

Los módulos EJB se depositan en META-INF/ejb-jar.xml y en WEB-INF/ejb-jar.xml para los módulos Web. EJB lite puede empaquetarse directamente en un archivo .war (Web Archive File) o .jar (Java Archive File).

Si tus requerimientos utilizan la especificación completa de EJBs (llamadas remotas, JMS, llamadas asíncronas, Web Services, etc), entonces se debe empaquetar en un archivo .jar y no en un archivo .war.

Un archivo .ear (Enterprise Archive File) es utilizado para empaquetar uno o más módulos, ya sean .jar o .war., en un archivo único, el cual es reconocido por el servidor de aplicaciones y éste se encarga de desplegar correctamente cada módulo empaquetado en el archivo .ear.

Como podemos observar en la figura, si necesitamos desplegar una aplicación Web, podemos empaquetar los EJBs y las clases de Entidad en archivos .jar separados, los Servlets, y páginas JSP o JSF dentro del archivo .war, y estos archivos agregarlos a un archivo .ear, el cual empaqueta todos los componentes en uno solo.

Desde la especificación EJB 3.1, el concepto de EJB Lite puede empaquetar componentes EJB directamente en un archivo .war, sin necesidad del archivo .jar.

Videotrainning >
Lección 2. Enterprise JavaBeans

Contenedor Embebido Java EE

- Un contenedor embebido tiene como finalidad proveer un ambiente de ejecución Java EE.



```

public void iniciarContenedorEJB() throws Exception {
    EJBContainer contenedor = EJBContainer.createEJBContainer();
    Context contexto = contenedor.getContext();
    HolaMundoEJB ejb = (HolaMundoEJB) contexto.lookup("java:global/classes/HolaMundoEJB");
    ejb.saluda();
}

```

Curso de Java EE
© Derechos Reservados Global Mentoring

En sus inicios los EJB para ser probados, debían desplegarse en un contenedor J2EE compatible, y hasta no haber sido desplegados no había forma de saber si un componente funcionaba o no.

Esto hacía muy lento el desarrollo de aplicaciones ya que el programador pasaba mucho tiempo desplegando su aplicación, únicamente para darse cuenta que debía corregir su código. Esto sin incluir el tiempo en detener y reiniciar el servidor de aplicaciones Java. Si una aplicación era de mediana a grande podía demorar varios minutos por cada cambio en un componente sólo para revisar si se había programado correctamente.

En la versión Java EE 6 y EJB 3.1 contamos con un contenedor embebido, el cual nos permite realizar pruebas unitarias de nuestros componentes empresariales. La idea del contenedor embebido es poder ejecutar componentes EJB dentro de aplicaciones Java SE (aplicaciones estándar), permitiendo utilizar la misma JVM (Java Virtual Machine) para ejecutar pruebas (testing), procesos de tipo batch, EJB en aplicaciones de escritorio, entre varias tareas más.

Un contenedor embebido provee del mismo ambiente de ejecución que un contenedor Java EE y puede manejar los mismos servicios: inyección de dependencias, acceso a componentes empresariales, acceso a CMT (Container-Managed Transactions) para el manejo de transacciones, etc.

En la figura podemos observar un ejemplo de cómo ejecutar el contenedor embebido, además utilizar JNDI para encontrar un EJB y ejecutar un método.

```

EJBContainer contenedor = EJBContainer.createEJBContainer();
Context contexto = contenedor.getContext();
HolaMundoEJB ejb = (HolaMundoEJB)
    contexto.lookup("java:global/classes/HolaMundoEJB");
ejb.saluda();

```




Ejercicio 3 y 4

- Abrir el documento PDF de Ejercicios del cursos de Java EE.
- Realizar las siguientes prácticas:
- **Ejercicio 3:** Creación de nuestro primer EJB y Testing.
- **Ejercicio 4:** Escribiendo un cliente de nuestro EJB.



En Global Mentoring promovemos la *Pasión por la Tecnología Java*.

Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados.

Además agregamos nuevos cursos para que continúes con tu preparación como consultor Java de manera profesional.

A continuación te presentamos nuestro listado de cursos en constante crecimiento:

- ✓ Fundamentos de Java
- ✓ Programación con Java
- ✓ Java con JDBC
- ✓ HTML, CSS y JavaScript
- ✓ Servlets y JSP's
- ✓ Struts Framework
- ✓ Hibernate Framework
- ✓ Spring Framework
- ✓ JavaServer Faces
- ✓ Java EE (EJB, JPA y Web Services)
- ✓ JBoss Administration

Datos de Contacto:

Sitio Web: www.globalmentoring.com.mx

Email: informes@globalmentoring.com.mx

Ayuda en Vivo: www.globalmentoring.com.mx/chat.html