

1 Introducción a la tecnología EJB

Los contenidos que vamos a ver en el tema son:

- Desarrollo basado en componentes
- Servicios proporcionados por el contenedor EJB
- Funcionamiento de componentes EJB
- Tipos de beans
- Desarrollo de beans
- Clientes de los beans
- Roles EJB
- Evolución de la especificación EJB
- Ventajas de la tecnología EJB

1.1 Desarrollo basado en componentes

Con la tecnología J2EE Enterprise JavaBeans es posible desarrollar componentes (*enterprise beans*) que luego puedes reutilizar y ensamblar en distintas aplicaciones que tengas que hacer para la empresa. Por ejemplo, podrías desarrollar un bean cliente que represente un cliente en una base de datos. Podrías usar después ese bean cliente en un programa de contabilidad o en una aplicación de comercio electrónico o virtualmente en cualquier programa en el que se necesite representar un cliente. De hecho, incluso sería posible que el desarrollador del bean y el ensamblador de la aplicación no fueran la misma persona, o ni siquiera trabajaran en la misma empresa.

El desarrollo basado en componentes promete un paso más en el camino de la programación orientada a objetos. Con la programación orientada a objetos puedes reutilizar clases, pero con componentes es posible reutilizar a mayor nivel de funcionalidades e incluso es posible modificar estas funcionalidades y adaptarlas a cada entorno de trabajo particular sin tocar el código del componente desarrollado. Aunque veremos el tema con mucho más detalle, en este momento puedes ver un componente como un objeto tradicional con un conjunto de servicios adicionales soportados en tiempo de ejecución por el contenedor de componentes. El contenedor de componentes se denomina *contenedor EJB* y es algo así como el sistema operativo en el que éstos residen. Recuerda que en Java existe un modelo de programación de objetos remotos denominado RMI. Con RMI es posible enviar peticiones a objetos que están ejecutándose en otra máquina virtual Java. Podemos ver un componente EJB como un objeto remoto RMI que reside en un contenedor EJB que le proporciona un conjunto de servicios adicionales.

Cuando estés trabajando con componentes tendrás que dedicarle tanta atención al despliegue (*deployment*) del componente como a su desarrollo. Entendemos por despliegue la incorporación del componente a nuestro contenedor EJB y a nuestro entorno de trabajo (bases de datos, arquitectura de la aplicación, etc.). El despliegue se define de forma declarativa, mediante un fichero XML (descriptor del despliegue, *deployment descriptor*) en el que se definen todas las características del bean.

El desarrollo basado en componentes ha creado expectativas sobre la aparición de una serie de empresas dedicadas a implementar y vender componentes específicos a terceros. Este mercado de componentes nunca ha llegado a tener la suficiente masa crítica como para crear una industria sostenible. Esto es debido a distintas razones, como la dificultad en el diseño de componentes genéricos capaces de adaptarse a distintos dominios de aplicación, la falta de estandarización de los dominios de aplicación o la diversidad de estos dominios. Aun así, existe un campo

creciente de negocio en esta área (puedes ver, como ejemplo, www.componentsource.com).

1.2 Servicios proporcionados por el contenedor EJB

En el apartado anterior hemos comentado que la diferencia fundamental entre los componentes y los objetos clásicos reside en que los componentes *viven* en un contenedor EJB que los envuelve proporcionando una capa de servicios añadidos. ¿Cuáles son estos servicios? Los más importantes son los siguientes:

- Manejo de transacciones: apertura y cierre de transacciones asociadas a las llamadas a los métodos del bean.
- Seguridad: comprobación de permisos de acceso a los métodos del bean.
- Concurrencia: llamada simultánea a un mismo bean desde múltiples clientes.
- Servicios de red: comunicación entre el cliente y el bean en máquinas distintas.
- Gestión de recursos: gestión automática de múltiples recursos, como colas de mensajes, bases de datos o fuentes de datos en aplicaciones heredadas que no han sido traducidas a nuevos lenguajes/entornos y siguen usándose en la empresa.
- Persistencia: sincronización entre los datos del bean y tablas de una base de datos.
- Gestión de mensajes: manejo de Java Message Service (JMS).
- Escalabilidad: posibilidad de constituir clusters de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.
- Adaptación en tiempo de despliegue: posibilidad de modificación de todas estas características en el momento del despliegue del bean.

Pensad en lo complicado que sería programar una clase "a mano" que implementara todas estas características. Como se suele decir, la programación de EJB es sencilla si la comparamos con lo que habría que implementar de hacerlo todo por uno mismo. Evidentemente, si en la aplicación que estás desarrollando no vas a necesitar estos servicios y va a tener un interfaz web, podrías utilizar simplemente páginas JSP y JDBC.

1.3 Funcionamiento de los componentes EJB

El funcionamiento de los componentes EJB se basa fundamentalmente en el trabajo del contenedor EJB. El contenedor EJB es un programa Java que corre en el servidor y que contiene todas las clases y objetos necesarios para el correcto funcionamiento de los enterprise beans.

En la figura siguiente puedes ver una representación de muy alto nivel del funcionamiento básico de los enterprise beans. En primer lugar, puedes ver que el cliente que realiza peticiones al bean y el servidor que contiene el bean están ejecutándose en máquinas virtuales Java distintas. Incluso pueden estar en distintos hosts. Otra cosa a resaltar es que el cliente *nunca* se comunica directamente con el enterprise bean, sino que el contenedor EJB proporciona un *EJBObject* que hace de interfaz. Cualquier petición del cliente (una llamada a un *método de negocio* del enterprise bean) se debe hacer a través del objeto EJB, el cual solicita al contenedor EJB una serie de servicios y se comunica con el enterprise bean. Por último, el bean realiza las peticiones correspondientes a la base de datos.

El contenedor EJB se preocupa de cuestiones como:

- ¿Tiene el cliente permiso para llamar al método?
- Hay que abrir la transacción al comienzo de la llamada y cerrarla al terminar.

- ¿Es necesario refrescar el bean con los datos de la base de datos?

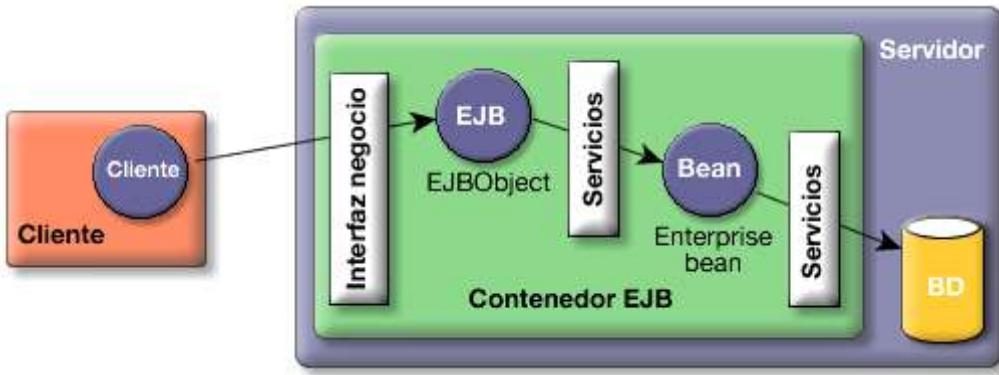


Figura 1.1: representación de alto nivel del funcionamiento de los enterprise beans.

Vamos a ver un ejemplo para que puedas entender mejor el flujo de llamadas. Supongamos que tenemos una aplicación de bolsa y el bean proporciona una implementación de un Broker. La interfaz de negocio del Broker está compuesta de varios métodos, entre ellos, por ejemplo, los métodos compra o vende. Supongamos que desde el objeto cliente queremos llamar al método compra. Esto va a provocar la siguiente secuencia de llamadas:

1. *Cliente:* "Necesito realizar una petición de compra al bean Broker."
2. *EJBObject:* "Espera un momento, necesito comprobar tus permisos."
3. *Contenedor EJB:* "Sí, el cliente tiene permisos suficientes para llamar al método compra."
4. *Contenedor EJB:* "Necesito un bean Broker para realizar una operación de compra. Y no olvidéis comenzar la transacción en el momento de instanciaros."
5. *Pool de beans:* "A ver... ¿a quién de nosotros le toca esta vez?".
6. *Contenedor EJB:* "Ya tengo un bean Broker. Pásale la petición del cliente."

Por cierto, la idea de usar este tipo de diálogos para describir el funcionamiento de un proceso o una arquitectura de un sistema informático es de Kathy Sierra en sus libros *"Head First Java"* y *"Head First EJB"*. Se trata de un tipo de libros radicalmente distintos a los habituales manuales de Java que consiguen que realmente aprendas este lenguaje cuando los sigues. Échales un vistazo si tienes oportunidad.

1.4 Tipos de beans

La tecnología EJB define tres tipos de beans: beans de sesión, beans de entidad y beans dirigidos por mensajes.

Los **beans de entidad** representan un objeto concreto que tiene existencia en alguna base de datos de la empresa. Una instancia de un bean de entidad representa una fila en una tabla de la base de datos. Por ejemplo, podríamos considerar el bean Cliente, con una instancia del bean siendo Eva Martínez (ID# 342) y otra instancia Francisco Gómez (ID# 120).

Los **beans dirigidos por mensajes** pueden escuchar mensajes de un servicio de mensajes JMS. Los clientes de estos beans nunca los llaman directamente, sino que es necesario enviar un mensaje JMS para comunicarse con ellos. Los beans dirigidos por mensajes no necesitan objetos EJBObject porque los clientes no se comunican nunca con ellos directamente. Un ejemplo de bean dirigido por mensajes podría ser un bean ListenerNuevoCliente que se activara cada vez que se envía un mensaje comunicando que se ha dado de alta a un nuevo cliente.

Por último, un **bean de sesión** representa un proceso o una acción de negocio. Normalmente, cualquier llamada a un servicio del servidor debería comenzar con una llamada a un bean de sesión. Mientras que un bean de entidad representa una cosa que se puede representar con un nombre, al pensar en un bean de sesión deberías pensar en un verbo. Ejemplos de beans de sesión podrían ser un carrito de la compra de una aplicación de negocio electrónico o un sistema verificador de tarjetas de crédito.

Vamos a describir con algo más de detalle estos tipos de bean. Comenzamos con los beans de sesión para continuar con los de entidad y terminar con los dirigidos por mensajes.

1.4.1 Beans de sesión

Los beans de sesión representan sesiones interactivas con uno o más clientes. Los beans de sesión pueden mantener un estado, pero sólo durante el tiempo que el cliente interactúa con el bean. Esto significa que los beans de sesión no almacenan sus datos en una base de datos después de que el cliente termine el proceso. Por ello se suele decir que los beans de sesión no son persistentes.

A diferencia de los beans de entidad, los beans de sesión no se comparten entre más de un cliente, sino que existe una correspondencia uno-uno entre beans de sesión y clientes. Por esto, el contenedor EJB no necesita implementar mecanismos de manejo de concurrencia en el acceso a estos beans.

Existen dos tipos de beans de sesión: con estado y sin él.

Beans de sesión sin estado

Los beans de sesión sin estado no se modifican con las llamadas de los clientes. Los métodos que ponen a disposición de las aplicaciones clientes son llamadas que reciben datos y devuelven resultados, pero que no modifican internamente el estado del bean. Esta propiedad permite que el contenedor EJB pueda crear una reserva (*pool*) de instancias, todas ellas del mismo bean de sesión sin estado y asignar cualquier instancia a cualquier cliente. Incluso un único bean puede estar asignado a múltiples clientes, ya que la asignación sólo dura el tiempo de invocación del método solicitado por el cliente.

Una de las ventajas del uso de beans de sesión, frente al uso de clases Java u objetos RMI es que no es necesario escribir los métodos de los beans de sesión de una forma segura para threads (*thread-safe*), ya que el contenedor EJB se va a encargar de que nunca haya más de un thread accediendo al objeto. Para ello usa múltiples instancias del bean para responder a peticiones de los clientes.

Cuando un cliente invoca un método de un bean de sesión sin estado, el contenedor EJB obtiene una instancia de la reserva. Cualquier instancia servirá, ya que el bean no puede guardar ninguna información referida al cliente. Tan pronto como el método termina su ejecución, la instancia del bean está disponible para otros clientes. Esta propiedad hace que los beans de sesión sin estado sean muy escalables para un gran número de clientes. El contenedor EJB no tiene que mover sesiones de la memoria a un almacenamiento secundario para liberar recursos, simplemente puede obtener recursos y memoria destruyendo las instancias.

Los beans de sesión sin estado se usan en general para encapsular procesos de negocio, más que datos de negocio (tarea de los entity beans). Estos beans suelen recibir nombres como ServicioBroker o GestorContratos para dejar claro que proporcionan un conjunto de procesos relacionados con un dominio específico del negocio.

Es apropiado usar beans de sesión sin estado cuando una tarea no está ligada a un cliente específico. Por ejemplo, se podría usar un bean sin estado para enviar un e-mail que confirme un pedido on-line o calcular unas cuotas de un préstamo.

También puede usarse un bean de sesión sin estado como un puente de acceso a una base de datos o a un bean de entidad. En una arquitectura cliente-servidor, el bean de sesión podría proporcionar al interfaz de usuario del cliente los datos necesarios, así como modificar objetos de negocio (base de datos o bean de entidad) a petición de la interfaz. Este uso de los beans de sesión sin estado es muy frecuente y constituye el denominado patrón de diseño *session facade*.

Algunos ejemplos de bean de sesión sin estado podrían ser:

- Un componente que comprueba si un símbolo de compañía está disponible en el mercado de valores y devuelve la última cotización registrada.
- Un componente que calcula la cuota del seguro de un cliente, basándose en los datos que se le pasa del cliente.

Beans de sesión con estado

En un bean de sesión con estado, las *variables de instancia* del bean almacenan datos específicos obtenidos durante la conexión con el cliente. Cada bean de sesión con estado, por tanto, almacena el estado conversacional de un cliente que interactúa con el bean. Este estado conversacional se modifica conforme el cliente va realizando llamadas a los métodos de negocio del bean. El estado conversacional no se guarda cuando el cliente termina la sesión.

La interacción del cliente con el bean se divide en un conjunto de pasos. En cada paso se añade nueva información al estado del bean. Cada paso de interacción suele denominarse con nombres como `setNombre` o `setDireccion`, siendo `nombre` y `direccion` dos variables de instancia del bean.

Algunos ejemplos de beans de sesión con estado podrían ser:

- Un ejemplo típico es un carrito de la compra, en donde el cliente va guardando uno a uno los ítem que va comprando.
- Un enterprise bean que reserva un vuelo y alquila un coche en un sitio Web de una agencia de viajes.

El estado del bean persiste mientras que existe el bean. A diferencia de los beans de entidad, no existe ningún recurso exterior al contenedor EJB en el que se almacene este estado.

Debido a que el bean guarda el estado conversacional con un cliente determinado, no le es posible al contenedor crear un almacén de beans y compartirlos entre muchos clientes. Por ello, el manejo de beans de sesión con estado es más pesado que el de beans de sesión sin estado.

En general, se debería usar un bean de sesión con estado si se cumplen las siguientes circunstancias:

- El estado del bean representa la interacción entre el bean y un cliente específico.
- El bean necesita mantener información del cliente a lo largo de un conjunto de invocaciones de métodos.
- El bean hace de intermediario entre el cliente y otros componentes de la aplicación, presentando una vista simplificada al cliente.

1.4.2 Beans de entidad

Los beans de entidad modelan conceptos o datos de negocio que puede expresarse como nombres. Esto es una regla sencilla más que un requisito formal, pero ayuda a determinar cuándo un concepto de negocio puede ser implementado como un bean de entidad. Los beans de entidad representan "cosas": objetos del mundo real como hoteles, habitaciones, expedientes, estudiantes, y demás. Un bean de entidad puede representar incluso cosas abstractas como una reserva. Los beans de entidad describen tanto el estado como la conducta de objetos del mundo real y permiten a los desarrolladores encapsular las reglas de datos y de negocio asociadas con un concepto específico. Por ejemplo un bean de entidad `Estudiante` encapsula los datos y reglas de negocio asociadas a un estudiante. Esto hace posible manejar de forma consistente y segura los datos asociados a un concepto.

Los beans de entidad se corresponden con datos en un almacenamiento persistente (base de datos, sistema de ficheros, etc.). Las variables de instancia del bean representan los datos en las columnas de la base de datos. El contenedor debe sincronizar las variables de instancia del bean con la base de datos. Los beans de entidad se diferencian de los beans de sesión en que las variables de instancia se almacenan de forma persistente.

Aunque entraremos en detalle más adelante, es interesante adelantar que el uso de los beans de entidad desde un cliente conlleva los siguientes pasos:

1. Primero el cliente debe obtener una referencia a la instancia concreta del bean de entidad que se está buscando (el estudiante "Francisco López") mediante un método `finder`. Estos métodos `finder` se encuentran definidos en la interfaz `home` e implementados en la clase `bean`. Los métodos `finder` pueden devolver uno o varios beans de entidad.
2. El cliente interactúa con la instancia del bean usando sus métodos `get` y `set`. El estado del bean se carga de la base de datos antes de procesar las llamadas a los métodos. Esto se encarga de hacerlo el contenedor de forma automática o el propio bean en la función `ejbLoad()`.
3. Por último, cuando el cliente termina la interacción con la instancia del bean sus contenidos se vuelcan en el almacen persistente. O bien lo hace de forma automática el contenedor o bien éste llama al método `ejbStore()`.

Son muchas las ventajas de usar beans de entidad en lugar de acceder a la base de datos directamente. El uso de beans de entidad nos da una perspectiva orientada a objetos de los datos y proporciona a los programadores un mecanismo más simple para acceder y modificar los datos. Es mucho más fácil, por ejemplo, cambiar el nombre de un estudiante llamando a `student.setName()` que ejecutando un comando SQL contra la base de datos. Además, el uso de objetos favorece la reutilización del software. Una vez que un bean de entidad se ha definido, su definición puede usarse a lo largo de todo el sistema de forma consistente. Un bean `Estudiante` proporciona una forma completa de acceder a la información del estudiante y eso asegura que el acceso a la información es consistente y simple.

La representación de los datos como beans de entidad puede hacer que el desarrollo sea más sencillo y menos costoso.

Diferencias con los beans de sesión

Los beans de entidad se diferencian de los beans de sesión, principalmente, en que son persistentes, permiten el acceso compartido, tienen clave primaria y pueden participar en relaciones con otros beans de entidad:

Persistencia

Debido a que un bean de entidad se guarda en un mecanismo de almacenamiento se dice que es persistente. Persistente significa que el estado del bean de entidad existe más tiempo que la duración de la aplicación o del proceso del servidor J2EE. Un ejemplo de datos persistentes son los datos que se almacenan en una base de datos.

Los beans de entidad tienen dos tipos de persistencia: **Persistencia Gestionada por el Bean** (BMP, *Bean-Managed Persistence*) y **Persistencia Gestionada por el Contenedor** (CMP, *Container-Managed Persistence*). En el primer caso (BMP) el bean de entidad contiene el código que accede a la base de datos. En el segundo caso (CMP) la relación entre las columnas de la base de datos y el bean se describe en el fichero de propiedades del bean, y el contenedor EJB se ocupa de la implementación.

Acceso compartido

Los clientes pueden compartir beans de entidad, con lo que el contenedor EJB debe gestionar el acceso concurrente a los mismos y por ello debe usar transacciones. La forma de hacerlo dependerá de la política que se especifique en los descriptores del bean.

Clave primaria

Cada bean de entidad tiene un identificador único. Un bean de entidad alumno, por ejemplo, puede identificarse por su número de expediente. Este identificador único, o *clave primaria*, permite al cliente localizar a un bean de entidad particular.

Relaciones

De la misma forma que una tabla en una base de datos relacional, un bean de entidad puede estar relacionado con otros EJB. Por ejemplo, en una aplicación de gestión administrativa de una universidad, el bean `alumnoEjb` y el bean `actaEjb` estarían relacionados porque un alumno aparece en un acta con una calificación determinada.

Las relaciones se implementan de forma distinta según se esté usando la persistencia manejada por el bean o por el contenedor. En el primer caso, al igual que la persistencia, el desarrollador debe programar y gestionar las relaciones. En el segundo caso es el contenedor el que se hace cargo de la gestión de las relaciones. Por ello, estas últimas se denominan a veces relaciones gestionadas por el contenedor.

1.4.3 Beans dirigidos por mensajes

Son el tercer tipo de beans propuestos por la última especificación de EJB. Estos beans permiten que las aplicaciones J2EE reciban mensajes JMS de forma asíncrona. Así, el hilo de ejecución de un cliente no se bloquea cuando está esperando que se complete algún método de negocio de otro enterprise bean. Los mensajes pueden enviarse desde cualquier componente J2EE (una aplicación cliente, otro enterprise bean, o un componente Web) o por una aplicación o sistema JMS que no use la tecnología J2EE.

Diferencias con los beans de sesión y de entidad

La diferencia más visible es que los clientes no acceden a los beans dirigidos por mensajes mediante interfaces (explicaremos esto con más detalle más adelante),

sino que un bean dirigido por mensajes sólo tienen una clase bean.

En muchos aspectos, un bean dirigido por mensajes es parecido a un bean de sesión sin estado.

- Las instancias de un bean dirigido por mensajes no almacenan ningún estado conversacional ni datos de clientes.
- Todas las instancias de los beans dirigidos por mensajes son equivalentes, lo que permite al contenedor EJB asignar un mensaje a cualquier instancia. El contenedor puede almacenar estas instancias para permitir que los streams de mensajes sean procesados de forma concurrente.
- Un único bean dirigido por mensajes puede procesar mensajes de múltiples clientes.

Las variables de instancia de estos beans pueden contener algún estado referido al manejo de los mensajes de los clientes. Por ejemplo, pueden contener una conexión JMS, una conexión de base de datos o una referencia a un objeto enterprise bean.

Cuando llega un mensaje, el contenedor llama al método `onMessage` del bean. El método `onMessage` suele realizar un casting del mensaje a uno de los cinco tipos de mensajes de JMS y manejarlo de forma acorde con la lógica de negocio de la aplicación. El método `onMessage` puede llamar a métodos auxiliares, o puede invocar a un bean de sesión o de entidad para procesar la información del mensaje o para almacenarlo en una base de datos.

Un mensaje puede enviarse a un bean dirigido por mensajes dentro de un contexto de transacción, por lo que todas las operaciones dentro del método `onMessage` son parte de un única transacción.

1.5 Desarrollo de beans

El desarrollo y programación de los beans suele ser un proceso bastante similar sea cual sea el tipo de bean. Consta de los siguientes 5 pasos:

1. Escribe y compila la clase *bean* que contiene a todos los métodos de negocio.
2. Escribe y compila las dos interfaces del bean: *home* y *componente*.
3. Crea un descriptor XML del despliegue en el que se describa qué es el bean y cómo debe manejarse. Este fichero debe llamarse `ejb-jar.xml`.
4. Pon la clase bean, los interfaces y el descriptor XML del despliegue en un fichero EJB JAR . Podría haber más de un bean el mismo fichero EJB JAR, pero nunca habrá más de un descriptor de despliegue.
5. Despliega el bean en el servidor usando las herramientas proporcionadas por el servidor de aplicaciones.

Vamos a ver con algo más de detalle estos cinco pasos, usando un ejemplo sencillo de un bean de sesión sin estado que implementa el método de negocio `saluda()` que devuelve un string con un saludo.

1.5.1 Clase bean

En la clase bean se encuentran los denominados métodos de negocio. Son los métodos finales a los que el cliente quiere acceder y los que debes programar. Son también los métodos definidos en la interfaz componente.

Lo primero que debes hacer es decidir qué tipo de bean necesitas implementar: un bean de sesión, de entidad o uno dirigido por mensajes. Estos tres tipos se definen con tres interfaces distintas: `SessionBean`, `EntityBean` y `MessageBean`. La clase bean que vas a escribir debe implementar una de ellas. En nuestro caso, vamos a definir un

bean de sesión sin estado, por lo que la clase SaludoBean implementará la interfaz SessionBean.

```
package especialista;
import javax.ejb.*;

public class SaludoBean implements SessionBean {
    private String[] saludos = {"Hola", "Que tal?", "Como estas?", "Cuanto tiempo sin verte!", "Que te cuentas?", "Que hay de nuevo?"};

    // Los cuatro métodos siguientes son los de la interfaz
    // SessionBean
    public void ejbActivate() {
        System.out.println("ejb activate");
    }

    public void ejbPassivate() {
        System.out.println("ejb pasivate");
    }

    public void ejbRemove() {
        System.out.println("ejb remove");
    }

    public void setSessionContext(SessionContext cntx) {
        System.out.println("set session context");
    }

    // El siguiente es el método de negocio, al que
    // van a poder llamar los clientes del bean
    public String saluda() {
        System.out.println("estoy en saluda");
        int random = (int) (Math.random() * saludos.length);
        return saludos[random];
    }

    // Por último, el método ejbCreate que no es de
    // la interfaz sessionBean sino que corresponde al
    // método de creación de beans de la interfaz Home del EJB
    public void ejbCreate() {
        System.out.println("ejb create");
    }
}
```

Pregunta:

- ¿Por qué los métodos de la clase bean, a diferencia de los métodos de las interfaces *componente* y *home*, no definen la excepción `RemoteException`?

1.5.2 Interfaces *componente* y *home*

Una vez implementado el fichero `SaludoBean.java`, en el que se define el enterprise bean, debes pasar a definir las interfaces *componente* y *home* del bean. Vamos a llamar a estas interfaces `Saludo` y `SaludoHome`. Estas interfaces son las que deberá usar el cliente para comunicarse con el bean.

La interfaz *componente* hereda de la interfaz `EJBObject` y en ella se definen los métodos de negocio del bean, los que va a poder llamar el cliente para pedir al bean que realice sus funciones:

```
package especialista;

import javax.ejb.*;
import java.rmi.RemoteException;
```

```
public interface Saludo extends EJBObject {
    public String saluda() throws RemoteException;
}
```

Todos los métodos definidos en esta interfaz se corresponden con los métodos de negocio del bean y todos van a ser métodos remotos (¿recuerdas RMI?), ya que van a implementarse en una máquina virtual Java distinta de la máquina. Por ello, todos estos métodos deben declarar la excepción `RemoteException`.

La interfaz `home` hereda de la interfaz `EJBHome`. El cliente usa los métodos de esta interfaz para obtener una referencia a la interfaz componente. Puedes pensar en el `home` como en una especie de fábrica que construye referencias a los beans y las distribuye entre los clientes.

```
package especialista;

import javax.ejb.*;
import java.rmi.RemoteException;

public interface SaludoHome extends EJBHome {
    public Saludo create() throws CreateException, RemoteException;
}
```

El método `create()` se corresponde con el método `ejbCreate()` definido en la clase `SaludoBean`, y debe devolver el tipo `Saludo` de la interfaz *componente*. La interfaz también va a ser una interfaz remota y, por tanto, debe declarar la excepción `RemoteException`. Además, el método `create` debe declarar la excepción `CreateException`.

Cuando se despliega un bean en el contenedor EJB, éste crea dos objetos que llamaremos `EJBObject` y `EJBHome` que implementarán estas interfaces. Estos objetos separan el bean del cliente, de forma que el cliente nunca accede directamente al bean. Así el contenedor puede incorporar sus servicios a los métodos de negocio.

Preguntas:

- ¿Dónde se deberán instalar los ficheros `.class` resultantes de las compilaciones de estas interfaces: en el servidor, en el cliente o en ambos?
- ¿Qué sucede si definimos algún método en la clase bean que después no lo definimos en la interfaz *componente*?

1.5.3 Descriptor del despliegue

Los tres ficheros Java anteriores son todo lo que tienes que escribir en Java. Recuerda: una clase (`SaludoBean`) y dos interfaces (`SaludoHome` y `Saludo`). Ya queda poco para terminar. El cuarto y último elemento es tan importante como los anteriores. Se trata del descriptor de despliegue (*deployment descriptor*, DD) del bean. El descriptor de despliegue es un fichero XML en el que se detalla todo lo que el servidor necesita saber para gestionar el bean. El nombre de este fichero siempre debe ser `ejb-jar.xml`.

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

<!DOCTYPE ejb-jar PUBLIC
'//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>

<ejb-jar>
    <display-name>Ejb1</display-name>
    <enterprise-beans>
```

```

<session>
    <display-name>SaludoBean</display-name>
    <ejb-name>SaludoBean</ejb-name>
    <home>especialista.SaludoHome</home>
    <remote>especialista.Saludo</remote>
    <ejb-class>especialista.SaludoBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
</session>

</enterprise-beans>
</ejb-jar>

```

En este caso, dado que el ejemplo es muy sencillo, tenemos que definir pocos elementos. Le estamos diciendo al servidor cuáles son las clases bean y las interfaces home y componente (en el fichero XML la llaman *remote*) del bean. Debes saber que no existe ninguna norma en la arquitectura EJB sobre los nombres de las distintas clases Java que conforman el bean. Por eso es necesario indicarle al servidor cuáles son éstas mediante los correspondientes elementos en el DD. También le decimos qué tipo de bean queremos utilizar (un bean se sesión sin estado).

Existen muchos más elementos que podemos definir en el DD. Los iremos viendo poco a poco a lo largo de las siguientes sesiones.

1.5.4 Fichero ejb-jar

Una vez escritas las clases e interfaces y el descriptor del despliegue, debemos compactar todos los ficheros resultantes (los ficheros .class y el fichero XML) en un único fichero JAR.

La estructura de este fichero JAR es:

```

/META-INF/ejb-jar.xml
/especialista/Saludo.class
/especialista/SaludoHome.class
/especialista/SaludoBean.class

```

En el directorio META-INF se incluye el DD. El resto del fichero JAR corresponde al directorio definido por el package especialista y a los tres ficheros .class que definen el bean.

La mayoría de servidores de aplicaciones proporcionan herramientas gráficas que crean el fichero de descripción y empaquetan el bean de forma automática.

Este fichero es el que se desplegará en el servidor de aplicaciones. Puedes nombrar a este fichero con el nombre que quieras. Una costumbre bastante usada es llamarlo <nOMBRE>-ejb.jar, siendo <nOMBRE> el nombre del bean o de la aplicación. En nuestro caso, podríamos llamarlo saludo-ejb.jar.

1.5.5 Despliegue del bean

Una vez construido el fichero EJB JAR es necesario desplegarlo en el servidor de aplicaciones. El despliegue conlleva dos acciones: la primera es darle al bean un nombre externo (un nombre JNDI, hablando más técnicamente) para que los clientes y otros beans puedan referirse a él. En nuestro caso, le daremos como nombre "SaludoBean". La segunda acción es la de llevar al servidor de aplicaciones el fichero EJB JAR.

Existen dos escenarios diferenciados para la realización del despliegue, dependiendo de si has desarrollado el bean en el mismo host que se encuentra el servidor de aplicaciones o en un host distinto. El primer caso suele suceder cuando estás trabajando en modo de prueba y estás depurando el desarrollo. El segundo caso suele suceder cuando ya has depurado el bean y quieres desplegarlo en modo de producción: ¡es recomendable no desarrollar y depurar en el mismo host en el que se encuentra el servidor de aplicaciones en producción!.

El proceso de despliegue no está definido en la especificación J2EE y cada servidor de aplicaciones tiene unas características propias. En general, la mayoría de servidores de aplicaciones proporcionan un interfaz gráfico de administración para gestionar el despliegue. También la mayoría de servidores proporcionan una tarea de ant para poder realizar el despliegue usando esta herramienta desde la línea de comando.

En la sesión práctica veremos un ejemplo de cada tipo con el servidor de aplicaciones weblogic de BEA.

Por último, hay un elemento previo al despliegue que, por simplificar, no hemos comentado. Se trata del proceso de ensamblaje de la aplicación (Application Assembly). En este proceso se construye, a partir de uno o más ficheros EJB JAR con beans, un único fichero de aplicación EAR en el que además se pueden asignar valores a ciertas constantes que serán utilizadas por los beans. Todo ello se hace sin necesidad de recompilar las clases e interfaces de cada uno de los beans.

1.6 Clientes de los beans

Una vez que el bean está desplegado en el contenedor, ya podemos usarlo desde un cliente. El cliente puede ser una clase Java cualquiera, ya sea un cliente aislado o un servlet que se está ejecutando en el contenedor web del servidor de aplicaciones. El código que deben ejecutar los clientes del bean es básicamente el mismo en cualquier caso.

Puedes ver a continuación el código de un cliente que usa el bean.

```

1. import java.io.*;
2. import java.text.*;
3. import java.util.*;
4. import javax.servlet.*;
5. import javax.servlet.http.*;
6. import javax.naming.*;
7. import javax.rmi.*;
8. import especialista.*;
9.
10. public class SaludoClient {
11.
12.     public static void main(String [] args) {
13.         try {
14.             Context jndiContext = getInitialContext();
15.             Object ref = jndiContext.lookup("SaludoBean");
16.             SaludoHome home = (SaludoHome)
17.                 PortableRemoteObject.narrow(ref, SaludoHome.class);
18.             Saludo sal = (Saludo)
19.                 PortableRemoteObject.narrow(home.create(), Saludo.class);
20.             System.out.println("Voy a llamar al bean");
21.             System.out.println(sal.saluda());
22.             System.out.println("Ya he llamado al bean");
23.         } catch (Exception e) {e.printStackTrace();}
24.     }
25.
26.     public static Context getInitialContext()
27.         throws javax.naming.NamingException {

```

```

28.     Properties p = new Properties();
29.     p.put(Context.INITIAL_CONTEXT_FACTORY,
30.             "weblogic.jndi.WLInitialContextFactory");
31.     p.put(Context.PROVIDER_URL, "t3://localhost:7001");
32.     return new javax.naming.InitialContext(p);
33. }
34. }
```

Básicamente, el cliente debe realizar siempre las siguientes tareas:

1. Acceder al servicio JNDI (línea 14 y líneas 26-34), obteniendo el contexto JNDI inicial. Para ello se llama a la función `javax.naming.InitialContext()`, pasándole como argumento unas propiedades dependientes del servidor que implementa el JNDI. En este caso estamos asumiendo que el servicio JNDI lo proporciona un servidor de aplicaciones BEA weblogic que está ejecutándose en el localhost.
2. Localizar el bean proporcionando a JNDI su nombre lógico (línea 15). En este caso, el nombre JNDI del bean es `saludoBean`.
3. Hacer un casting del objeto que devuelve JNDI para convertirlo en un objeto de la clase `SaludoHome` (líneas 16 y 17). La forma de hacer el casting es especial, ya que antes de hacer el casting hay que obtener un objeto Java llamando al método `PortableRemoteObject.narrow()` porque estamos recibiendo de JNDI un objeto que ha sido serializado usando el protocolo IIOP.
4. Llamar al método `create()` del objeto home para crear un objeto de tipo `saludo` (línea 19). Lo que se obtiene es un stub (ya hablaremos más de ello en la siguiente sesión) y hay que llamar otra vez a `narrow` para asegurarse de que el objeto devuelto satisface
5. Llamar a los métodos de negocio del bean (línea 21).

1.7 Roles EJB

La arquitectura EJB define seis papeles principales. Brevemente, son:

- **Desarrollador de beans:** desarrolla los componentes enterprise beans.
- **Ensamblador de aplicaciones:** compone los enterprise beans y las aplicaciones cliente para conformar una aplicación completa
- **Desplegador:** despliega la aplicación en un entorno operacional particular (servidor de aplicaciones)
- **Administrador del sistema:** configura y administra la infraestructura de computación y de red del negocio
- **Proporcionador del Contenedor EJB y Proporcionador del Servidor EJB:** un fabricante (o fabricantes) especializado en manejo de transacciones y de aplicaciones y otros servicios de bajo nivel. Desarrollan el servidor de aplicaciones.

1.8 Evolución de la especificación EJB

En Marzo de 1998 Sun Microsystems propone la especificación 1.0 de la arquitectura Enterprise JavaBeans. Esta especificación comienza con la siguiente definición:

La arquitectura Enterprise JavaBeans es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones de empresa distribuidas y orientadas a objetos. Las aplicaciones escritas usando la arquitectura Enterprise JavaBeans son escalables, transaccionales y seguras para multi usuarios. Estas aplicaciones pueden escribirse una vez, y luego desplegarse en cualquier servidor que soporte la especificación Enterprise JavaBeans.

Aunque se han introducido nuevas versiones de la especificación, que incorporan muchas mejoras a la propuesta inicial, la definición de la arquitectura sigue siendo la misma. La siguiente tabla muestra las distintas revisiones que ha sufrido la especificación de la arquitectura EJB.

Especificación EJB	Fecha	Principales novedades
EJB 1.0	Marzo 1998	Propuesta inicial de la arquitectura EJB. Se introducen los beans de sesión y los de entidad (de implementación opcional). Persistencia manejada por el contenedor en los beans de entidad. Manejo de transacciones. Manejo de seguridad.
EJB 1.1	Diciembre 1999	Implementación obligatoria de los beans de entidad. Acceso al entorno de los beans mediante JNDI.
EJB 2.0	Agosto 2001	Manejo de mensajes con los beans dirigidos por mensajes. Relaciones entre beans manejadas por el contenedor. Uso de interfaces locales entre beans que se encuentran en el mismo servidor. Consultas de beans declarativas, usando el EJB QL.
EJB 2.1	Agosto 2002	Soporte para servicios web. Temporizador manejado por el contenedor de beans. Mejora en el EJB QL.

Tabla 1: Evolución de las especificaciones de la arquitectura Enterprise JavaBeans

1.9 Ventajas de la tecnología EJB

La arquitectura EJB proporciona beneficios a todos los papeles que hemos mencionado previamente (desarrolladores, ensambladores de aplicaciones, administradores, desplegadores, fabricantes de servidores). Vamos a enumerar las ventajas que obtendrán los desarrolladores de aplicaciones y los clientes finales.

Las ventajas que ofrece la arquitectura Enterprise JavaBeans a un desarrollador de aplicaciones se listan a continuación.

- **Simplicidad.** Debido a que el contenedor de aplicaciones libera al programador de realizar las tareas del nivel del sistema, la escritura de un enterprise bean es casi tan sencilla como la escritura de una clase Java. El desarrollador no tiene que preocuparse de temas de nivel de sistema como la seguridad, transacciones, multi-threading o la programación distribuida. Como

- resultado, el desarrollador de aplicaciones se concentra en la lógica de negocio y en el dominio específico de la aplicación.
- **Portabilidad de la aplicación.** Una aplicación EJB puede ser desplegada en cualquier servidor de aplicaciones que soporte J2EE.
 - **Reusabilidad de componentes.** Una aplicación EJB está formada por componentes enterprise beans. Cada enterprise bean es un bloque de construcción reusable. Hay dos formas esenciales de reusar un enterprise bean a nivel de desarrollo y a nivel de aplicación cliente. Un bean desarrollado puede desplegarse en distintas aplicaciones, adaptando sus características a las necesidades de las mismas. También un bean desplegado puede ser usado por múltiples aplicaciones cliente.
 - **Possibilidad de construcción de aplicaciones complejas.** La arquitectura EJB simplifica la construcción de aplicaciones complejas. Al estar basada en componentes y en un conjunto claro y bien establecido de interfaces, se facilita el desarrollo en equipo de la aplicación.
 - **Separación de la lógica de presentación de la lógica de negocio.** Un enterprise bean encapsula típicamente un proceso o una entidad de negocio. (un objeto que representa datos del negocio), haciéndolo independiente de la lógica de presentación. El programador de gestión no necesita de preocuparse de cómo formatear la salida; será el programador que desarrolle la página Web el que se ocupe de ello usando los datos de salida que proporcionará el bean. Esta separación hace posible desarrollar distintas lógicas de presentación para la misma lógica de negocio o cambiar la lógica de presentación sin modificar el código que implementa el proceso de negocio.
 - **Despliegue en muchos entornos operativos.** Entendemos por entornos operativos el conjunto de aplicaciones y sistemas (bases de datos, sistemas operativos, aplicaciones ya en marcha, etc.) que están instaladas en una empresa. Al detallarse claramente todas las posibilidades de despliegue de las aplicaciones, se facilita el desarrollo de herramientas que asistan y automaticen este proceso. La arquitectura permite que los beans de entidad se conecten a distintos tipos de sistemas de bases de datos.
 - **Despliegue distribuido.** La arquitectura EJB hace posible que las aplicaciones se desplieguen de forma distribuida entre distintos servidores de una red. El desarrollador de beans no necesita considerar la topología del despliegue. Escribe el mismo código independientemente de si el bean se va a desplegar en una máquina o en otra (cuidado: con la especificación 2.0 esto se modifica ligeramente, al introducirse la posibilidad de los interfaces locales).
 - **Interoperabilidad entre aplicaciones.** La arquitectura EJB hace más fácil la integración de múltiples aplicaciones de diferentes vendedores. El interfaz del enterprise bean con el cliente sirve como un punto bien definido de integración entre aplicaciones.
 - **Integración con sistemas no-Java.** Las APIs relacionadas, como las especificaciones Connector y Java Message Service (JMS), así como los beans manejados por mensajes, hacen posible la integración de los enterprise beans con sistemas no Java, como sistemas ERP o aplicaciones mainframes.
 - **Recursos educativos y herramientas de desarrollo.** El hecho de que la especificación EJB sea un estándar hace que exista una creciente oferta de herramientas y formación que facilita el trabajo del desarrollador de aplicaciones EJB.

Entre las ventajas que aporta esta arquitectura al cliente final, destacamos la posibilidad de elección del servidor, la mejora en la gestión de las aplicaciones, la integración con las aplicaciones y datos ya existentes y la seguridad.

- **Elección del servidor.** Debido a que las aplicaciones EJB pueden ser ejecutadas en cualquier servidor J2EE, el cliente no queda ligado a un vendedor de servidores. Antes de que existiera la arquitectura EJB era muy difícil que una aplicación desarrollada para una determinada capa intermedia

(Tuxedo, por ejemplo) pudiera portarse a otro servidor. Con la arquitectura EJB, sin embargo, el cliente deja de estar atado a un vendedor y puede cambiar de servidor cuando sus necesidades de escalabilidad, integración, precio, seguridad, etc. lo requieran.

Existen en el mercado algunos servidores de aplicaciones gratuitos (JBoss, el servidor de aplicaciones de Sun, etc.) con los que sería posible hacer unas primeras pruebas del sistema, para después pasar a un servidor de aplicaciones con más funcionalidades.

- **Gestión de las aplicaciones.** Las aplicaciones son mucho más sencillas de manejar (arrancar, parar, configurar, etc.) debido a que existen herramientas de control más elaboradas.
- **Integración con aplicaciones y datos ya existentes.** La arquitectura EJB y otras APIs de J2EE simplifican y estandarizan la integración de aplicaciones EJB con aplicaciones no Java y sistemas en el entorno operativo del cliente. Por ejemplo, un cliente no tiene que cambiar un esquema de base de datos para encajar en una aplicación. En lugar de ello, se puede construir una aplicación EJB que encaje en el esquema cuando sea desplegada.
- **Seguridad.** La arquitectura EJB traslada la mayor parte de la responsabilidad de la seguridad de una aplicación al desarrollador de aplicaciones al vendedor del servidor, el Administrador de Sistemas y al Desplegador (papeles de la especificación EJB). La gente que ejecuta esos papeles están más cualificados que el desarrollador de aplicaciones para hacer segura la aplicación. Esto lleva a una mejor seguridad en las aplicaciones operacionales.