# EJB (Enterprise Java Beans)

## EJB

* **¿Qué es?** Es un componente del lado del servidor que encapsula la lógica de negocio de una aplicación.
* **¿Cuándo Usar un EJB?** Los EJB se utilizan cuando la aplicación tiene que ser escalable, las transacciones son necesarias para garantizar la integridad de los datos y existe una variedad de clientes.
* **Beneficios:** 
  + Simplifica el desarrollo de las aplicaciones distribuidas.
  + Como los EJBs (y no los clientes) contienen la lógica de negocio de la aplicación, el programador de la capa cliente puede dedicarse a mejorar la presentación hacia el cliente.
  + Los enterprise beans son componentes portables y pueden construirse nuevas aplicaciones ensamblando beans existentes.
* **Tipos de EJB:**
  + Session Beans
  + Message Driven Beans
  + Entity Beans

## Entity Beans (JPA)

Representa datos persistentes almacenados en una fila de una tabla/relación de una base de datos. Si el cliente termina o si se apaga el servidor, los servicios subyacentes se aseguran de grabar el bean.

## Message Driven Beans (JMS)

Combina las características de un bean de sesión y de un oyente de Java Message Service (JMS), permitiendo que un componente de negocio reciba asíncronamente mensajes JMS.

## Session Beans

* Sirven para encapsular procesos de negocio. Representan una conversación temporal con un cliente; cuando el cliente finaliza su ejecución, el bean de sesión y sus datos desaparecen.
* **Características**:
  + Comunicación sincrónica.
  + No son persistentes (sus datos no se guardan en una base de datos).
  + Son dedicados a un único cliente.
  + Tienen vida corta.
* **Estructura:** Un session bean está compuesto por una clase que implementa interfaces (POJO + anotaciones = EJB).
* **Tipos de Interfaces:**
  + **Interfaz Remota (@Remote):** 
    - Ejecuta en una JVM diferente al enterprise bean que accede.
    - Un cliente remoto puede ser un componente web, una aplicación cliente u otro enterprise bean.
    - Para el cliente remoto la ubicación del enterprise bean es transparente.
  + **Interfaz Local (@Local):**
    - Ejecuta en la misma JVM al enterprise bean que accede.
    - El cliente local puede ser un componente web u otro enterprise bean.
* **Tipos de Session Beans:**
  + **Session Bean Sin Estado (Stateless):** 
    - No mantiene estado conversacional.
    - Es asignado por el contenedor a cualquier cliente (pool de instancias).
    - Ofrecen mejor escalabilidad.
    - Pueden implementar servicios web (web service).
    - Ciclo de Vida:

Los métodos callback son métodos del bean que el contenedor llama para notificar la transición del ciclo de vida de un bean. Estos métodos son marcados con anotaciones como @PostConstruct y @PreDestroy. Son públicos, retornan void y no reciben parámetros.

* + **Session Bean Con Estado (StateFull):**
    - Matiene el estado conversacional.
    - La instancia es reservada para el cliente y cada una almacena la información del cliente.
    - La sesión finaliza si el cliente remueve el bean o finaliza su sesión.

## Acceso a Enterprise Beans

* **JNDI (Java Naming and Directory Interface)**: Es un servicio de nombres que permite a un componente localizar otros componentes o recursos.
  + Métodos:
    - Asociar un nombre con un objeto (binding).
    - Buscar un objeto (lookup).
  + JNDI provee un árbol (memoria) para el almacenamiento y búsqueda de los objetos (context). Todas las operaciones son realizadas en un contexto.
  + Initial Conext: Es el punto de partida para hacer las operaciones. Hay que pasar la información necesaria para obtener el contexto. Para ello se usa la clase Properties, o bien se puede utilizar un archivo de configuración (jndi.properties).
  + Propiedades del ambiente JNDI: Se deben configurar de acuerdo con las características del proveedor del servicio que se va a utilizar.
    - Java.naming.factory.initial: Nombre completo de la variable que representa la fábrica a utilizar para crear el contexto.
    - Java.naming.provider.url: Especifica la dirección de proveedor de servicios JNDI.
    - Java.naming.factory.url.pkgs: Se indican los paquetes (separados por “:”) utilizados para la carga de la fabrica de contexto.
* **Inyección de Dependencia** 
  + Se especializa en inyectar objetos en una clase, en lugar de ser la propia clase quien cree el objeto.
  + En EJB es el contenedor de EJBs quien inyecta en cada objeto los objetos necesarios según las relaciones plasmadas en las anotaciones.
  + Se utiliza la anotación @EJB.
  + El contenedor puede inicializar variables de instancia: Setter Injection o Constructor Injection.
  + Se pueden utilizar para: EJBContext, EntityManager, Resources, o referencias a session beans.

Con JNDI es responsabilidad del cliente hacer la localización y obtener la referencia al objeto.

Con EJB3, se puede utilizar la inyección de dependencia, dejando que el contenedor se responsabilice de inyectar un objeto basado en la declaración de la dependecia.

# Accessing Enterprise Beans

Una vista sin interfaz de un enterprise bean expone los métodos públicos de la clase de implementación de beans enterprise a clientes. Los clientes que utilizan la vista sin interfaz de un bean de empresa pueden invocar a cualquier método público de la clase de implementación de bean enterprise o cualquier superclases de la clase de implementación. Un negocio interfaceis una interfaz de lenguaje de programación Java estándar que contiene los métodos de negocio del bean enterprise. Un cliente puede acceder a un bean de sesión único a través de los métodos definidos en la interfaz de negocio del bean o por medio de los métodos públicos de un bean de empresa que tiene una vista sin interfaz. La interfaz de negocios o sin interfaz vista define la vista del cliente de un bean enterprise. Todos los demás aspectos de la empresa frijol (implementaciones de métodos y la configuración de implementación) están ocultos desde el cliente.

Las interfaces bien diseñadas y vistas sin interfaz simplifican el desarrollo y mantenimiento de aplicaciones Java EE. No sólo las interfaces limpias y sin interfaz vistas protegen los clientes de cualquier complejidad en el nivel EJB, sino que también permiten a los beans enterprise para cambiar internamente y sin afectar a los clientes. Por ejemplo, si cambia la implementación de un método de negocio bean de sesión, usted no tiene que alterar el código de cliente. Pero si tuviera que cambiar las definiciones de método en las interfaces, es posible que tenga que modificar el código de cliente también. Por lo tanto, es importante que el diseño de las interfaces y puntos de vista sin interfaz cuidadosamente para aislar a sus clientes de los posibles cambios en los beans enterprise. Los beans de sesión puede tener más de una interfaz de negocio. Los beans de sesión deben, pero no están obligados a, implementar su interfaz de negocio o interfaces.

## Using Enterprise Beans in Clients

El cliente de un bean empresarial obtiene una referencia a una instancia de un bean empresarial ya sea a través de la inyección de dependencia, utilizando Java anotaciones de lenguaje de programación, orJNDI búsqueda, utilizando la sintaxis de Java Naming and Directory Interface para encontrar la instancia del bean empresarial.

Inyección de dependencia es la forma más sencilla de obtener una referencia de enterprise bean. Los clientes que se ejecutan en un entorno gestionado por el servidor Java EE, JavaServer Faces aplicaciones web, JAX-RS servicios web, otros beans enterprise, o clientes de aplicaciones Java EE, soporta la inyección de dependencia mediante la anotación javax.ejb.EJB. Las aplicaciones que se ejecutan fuera de un entorno gestionado por servidor Java EE, como las aplicaciones Java SE, debe realizar una búsqueda explícita. JNDI soporta una sintaxis global para la identificación de los componentes de Java EE para simplificar esta búsqueda explícita.

## Portable JNDI Syntax

Three JNDI namespaces are used for portable JNDI lookups: java:global, java:module, and java:app.

**The Java: Global**: Espacio de nombres JNDI es la manera portátil de encontrar enterprise beans remotos utilizando búsquedas JNDI. Direcciones JNDI son de la siguiente forma:

java:global[/application name]/module name/enterprise bean name[/interface name]

Nombre de la aplicación y el nombre de módulo predeterminado para el nombre de la aplicación y el módulo menos la extensión de archivo. Nombres de las aplicaciones sólo son necesarios si la aplicación se empaqueta dentro de un EAR. El nombre de la interfaz sólo se requiere si el enterprise bean implementa más de una interfaz de negocio.

**The Java: Modulo.** El Namespace se utiliza para buscar beans empresariales locales dentro del mismo módulo. Direcciones JNDI utilizando el java: namespace módulo son de la siguiente forma:

java:module/enterprise bean name/[interface name]

El nombre de la interfaz sólo es necesario si el enterprise bean implementa más de un

interfaz de negocio.

**The Java: app.** El Namespace se utiliza para buscar beans empresariales locales envasados ​​dentro de la misma aplicación. Es decir, la empresa de frijol se empaqueta dentro de un archivo EAR que contiene múltiples módulos Java EE. Direcciones JNDI utilizando el java: namespace aplicación son de la siguiente forma:

java:app[/module name]/enterprise bean name[/interface name]

El nombre del módulo es opcional. El nombre de la interfaz sólo se requiere si el enterprise bean implementa más de una interfaz de negocio.

Por ejemplo, si un bean empresarial, MyBean, se envasa en el archivo de la aplicación web myApp.war, el nombre del módulo es myApp. The portable JNDI name is java:module/MyBean An equivalent JNDI name using the java:global namespace is java:global/myApp/MyBean.

## Decidir sobre acceso remoto o local

Cuando se diseña una aplicación Java EE, una de las primeras decisiones que usted toma es el tipo de acceso de cliente permitido por los beans enterprise:,, o servicio web local remoto.

Acoplamiento apretado o flojo de los granos relacionados: los granos de estructura rígida dependen el uno del otro.

Por ejemplo, si un bean de sesión que procesa los pedidos de ventas llama a un bean de sesión que los correos electrónicos un mensaje de confirmación al cliente, estos granos están estrechamente unida. Habas de estructura rígida son buenos candidatos para el acceso local. Debido a que encajan entre sí como una unidad lógica, que normalmente se llaman entre sí a menudo y se beneficiarían del aumento de rendimiento que es posible con el acceso local.

■ Tipo de cliente: Si se accede a un Enterprise Bean por clientes de la aplicación, se debe permitir el acceso remoto. En un entorno de producción, estos clientes casi siempre se ejecutan en máquinas distintas de aquellas en las que el GlassFish Server se está ejecutando. Si los clientes de una empresa de frijol son componentes web u otros beans de empresa, el tipo de acceso depende de cómo desea distribuir sus componentes.

■ distribución de componentes: las aplicaciones Java EE son escalables porque su lado servidor

componentes pueden ser distribuidos a través de múltiples máquinas. En una aplicación distribuida, por ejemplo, el servidor que los componentes se ejecutan en web puede no ser aquella en la que el

se despliegan beans empresariales que acceden. En este escenario distribuido, los beans empresariales deben permitir el acceso remoto.

■ Rendimiento: Debido a factores como la latencia de la red, las llamadas remotas puede ser más lento que las llamadas locales. Por otro lado, si usted distribuye componentes entre diferentes servidores, es posible mejorar el rendimiento general de la aplicación. Ambas declaraciones son

generalizaciones; el rendimiento puede variar en diferentes entornos operativos. Sin embargo, usted debe tener en cuenta cómo el diseño de su aplicación puede afectar al rendimiento. Si no está seguro de qué tipo de acceso de un bean enterprise debe tener, elija el acceso remoto. Esta decisión le da más flexibilidad. En el futuro, usted puede distribuir sus componentes para dar cabida a las crecientes demandas de su aplicación.

Aunque es infrecuente, es posible que un bean empresarial para permitir tanto el acceso remoto y local. Si este es el caso, ya sea la interfaz de negocio del bean debe ser designado explícitamente como una interfaz de negocios al ser condecorado con @Remote o @local, o la clase de bean debe designar explícitamente las interfaces de negocio mediante el uso de @Remote y Las anotaciones locales. La misma interfaz de negocio no puede ser a la vez un local y una interfaz de negocios a distancia.

## Clientes locales

Un cliente local tiene estas características.

■ Se debe ejecutar en la misma aplicación que el enterprise bean que accede.

■ Puede ser un componente web u otro bean enterprise.

■ Para el cliente local, la ubicación del enterprise bean que accede no es transparente.

La vista sin interfaz de un enterprise bean es una vista local. Los métodos públicos de la clase de implementación de bean enterprise se exponen a los clientes locales que acceden a la vista sin interfaz del bean enterprise. Beans Enterprise que utilizan la vista sin interfaz no implementan una interfaz de negocio.

La interfaz de negocios local define los métodos de negocio y de ciclo de vida del bean. Si interfaz de negocio del bean no está decorado conLocal oRemote, y si la clase del bean no especifica la interfaz usandoLocal oRemote, la interfaz de negocio es por defecto una interfaz local.

Para construir un bean empresarial que sólo permite el acceso local, usted puede, pero no están obligados a, siga uno de los siguientes:

■ Crear una clase de implementación de Enterprise Bean que no implementa un negocio interfaz, lo que indica que el bean expone una vista sin interfaz para los clientes. Por ejemplo:

@Session

public class MyBean { ... }

■ Anotar la interfaz de negocio de la empresa de frijol como una interfazLocal. Por ejemplo:

@Local

public interface InterfaceName { ... }

■ Especifique la interfaz mediante la decoración de la clase de bean conLocal y especifique el nombre de la interfaz. Por ejemplo:

@Local(InterfaceName.class)

public class BeanName implements InterfaceName { ... }

## Accessing Local Enterprise Beans Using the No-Interface View

Acceso de cliente a un enterprise bean que expone un local sin interfaz vista se logra a través de cualquiera de inyección de dependencia o búsqueda JNDI.

■ Para obtener una referencia a la vista sin interfaz de un bean empresarial a través de la inyección de dependencias, utilice la anotación javax.ejb.EJB y especificar la clase de implementación del bean enterprise:

@EJB

ExampleBean exampleBean;

■ Para obtener una referencia a la vista sin interfaz de un bean empresarial a través de búsqueda JNDI, utilice el método de búsqueda de la interfaz javax.naming.InitialContext:

ExampleBean exampleBean = (ExampleBean)

InitialContext.lookup("java:module/ExampleBean");

Los clientes no utilizan el nuevo operador para obtener una nueva instancia de un bean empresarial que utiliza una sin interfaz.

## Accessing Local Enterprise Beans That Implement Business Interfaces

Acceso clientes a enterprise beans que implementan las interfaces de negocios locales se lleva a cabo a través de cualquiera de inyección de dependencia o búsqueda JNDI.

■ Para obtener una referencia a la interfaz local de negocios de un bean empresarial a través de la inyección de dependencias, utilice la anotación javax.ejb.EJB y especificar nombre de la interfaz empresarial local del enterprise bean:

@EJB

Example example;

■ Para obtener una referencia a una interfaz de negocios local de un enterprise bean a través de búsqueda JNDI, utilice el método de búsqueda de la interfaz javax.naming.InitialContext:

ExampleLocal example = (ExampleLocal)

InitialContext.lookup("java:module/ExampleLocal");

## Clientes remotos

Un cliente remoto de un enterprise bean tiene los siguientes rasgos.

■ Se puede ejecutar en un equipo diferente y una JVM diferente del enterprise bean que accede. (No es necesario para ejecutar en una JVM distinta.)

■ Puede ser un componente web, un cliente de aplicación, u otro bean enterprise.

■ Para un cliente remoto, la ubicación del enterprise bean es transparente.

■ El enterprise bean debe implementar una interfaz de negocio. Es decir, los clientes remotos no pueden acceder a un bean empresarial a través de una vista sin interfaz.

Para crear un bean empresarial que permite el acceso remoto, debe

■ Decora la interfaz de negocio del bean enterprise con la anotación @ Remote:

@Remote

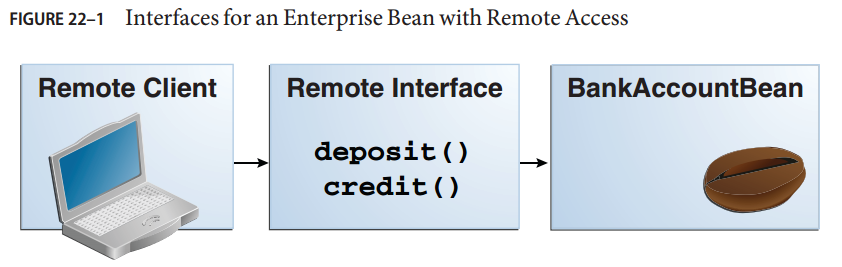
public interface InterfaceName { ... }

■ Decora la clase de bean con @ Remote, especificando la interfaz de negocio o interfaces:

@Remote(InterfaceName.class)

public class BeanName implements InterfaceName { ... }

El interface remoto define los métodos de negocio y de ciclo de vida que son específicos para el frijol. Por ejemplo, la interfaz remota de un bean llamado BankAccountBean podría tener métodos de negocio llamado depósito y de crédito. La Figura 22-1 muestra cómo la interfaz controla la vista del cliente de un bean enterprise.



Acceso de cliente a un enterprise bean que implementa una interfaz de negocios a distancia se logra a través de cualquiera de inyección de dependencia o búsqueda JNDI.

■ Para obtener una referencia a la interfaz de negocio remota de un bean empresarial a través de la inyección de dependencias, utilice la anotación javax.ejb.EJB y especificar nombre de la interfaz empresarial remota del bean enterprise:

@EJB

Example example;

■ Para obtener una referencia a una interfaz de negocio remota de un bean empresarial a través de búsqueda JNDI, utilice el método de búsqueda de la interfaz javax.naming.InitialContext:

ExampleRemote example = (ExampleRemote)

InitialContext.lookup("java:global/myApp/ExampleRemote");

## Web Servicies Clients

Un cliente de servicios web puede acceder a una aplicación Java EE de dos maneras. En primer lugar, el cliente puede acceder a un servicio web creado con JAX-WS. (Para obtener más información sobre JAX-WS, consulte el Capítulo 19, "Servicios Web de edificio con JAX-WS.") En segundo lugar, un cliente de servicios web puede invocar los métodos de negocio de un bean de sesión sin estado. Habas de mensajes no pueden ser accedidos por clientes de servicios web.

Siempre que utiliza los protocolos correctos (SOAP, HTTP, WSDL), cualquier cliente de servicios web puede acceder a un bean de sesión sin estado, si el cliente está escrito en el lenguaje de programación Java. El cliente no tiene ni siquiera "sabe" lo que la tecnología pone en práctica el servicio: bean de sesión sin, JAX-WS, o alguna otra tecnología. Además, Enterprise Beans y componentes web pueden ser clientes de servicios web. Esta flexibilidad le permite integrar aplicaciones Java EE con los servicios web.

Un cliente de servicios web accede a un bean de sesión sin estado a través del servicio web clase de implementación de punto final de la haba. Por defecto, todos los métodos públicos de la clase bean son accesibles para los clientes de servicios web. La @WebMethod se puede utilizar para personalizar el comportamiento de los métodos de servicio web. Si la @WebMethod se utiliza para decorar los métodos de la clase bean, sólo aquellos métodos decoradas conWebMethod están expuestos a clientes de servicios web.

## Method Parameters and Access

El tipo de acceso afecta a los parámetros de los métodos de frijol que se denominan por los clientes. En las siguientes secciones se aplican no sólo a los parámetros del método, sino también a los valores de retorno del método.

### Aislamiento

Los parámetros de las llamadas remotas están más aislados que los de llamadas locales. Con las llamadas remotas, el cliente y el frijol operan en diferentes copias de un objeto de parámetro. Si el cliente cambia el valor del objeto, el valor de la copia en el grano no cambia. Esta capa de aislamiento puede ayudar a proteger el grano si el cliente modifica accidentalmente los datos.

En una llamada local, el cliente y el frijol puede modificar el mismo objeto de parámetro. En general, usted no debe confiar en este efecto secundario de las llamadas locales. Tal vez algún día tendrá que distribuir sus componentes, en sustitución de las llamadas locales con los remotos.

Al igual que con los clientes remotos, clientes de servicios web operan en diferentes copias de parámetros que hace el bean que implementa el servicio web.

### La granularidad de acceso a los datos

Debido a que las llamadas remotas tienden a ser más lento que las llamadas locales, los parámetros en los métodos remotos deben ser relativamente de grano grueso. Un objeto genérico contiene más datos que una de grano fino, se requieren por lo menos llamadas de acceso. Por la misma razón, los parámetros de los métodos llamados por clientes de servicios web también deben ser de grano grueso.

# The Contents of an Enterprise Bean

Desarrollar un enterprise bean, debe proporcionar los siguientes archivos:

■ **clase enterprise bean**: Implementa los métodos de negocio del enterprise bean y los métodos de devolución de llamada de ciclo de vida.

■ **interfaces de Negocio**: Definir los métodos comerciales implementados por la clase bean enterprise. Una interfaz de negocio no es necesario si el enterprise bean expone una vista sin interfaz local.

■ **Clases de ayuda**: Otras clases que necesita la clase bean empresarial, tales como clases de excepción y de servicios públicos. Paquete de los artefactos de programación en la lista anterior, ya sea en un archivo JAR EJB (un módulo independiente que almacena el bean enterprise) o dentro de un archivo de la aplicación web (WAR)

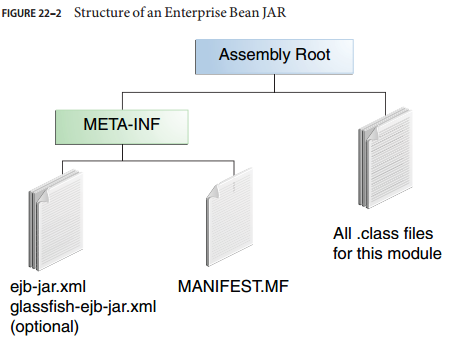
módulo.

## Packaging Enterprise Beans in EJB JAR Modules

Un archivo JAR EJB es portable y se puede utilizar para diversas aplicaciones.

Para el montaje de una aplicación, el paquete Java EE uno o más módulos, como archivos JAR EJB, en un archivo EAR, el archivo de almacenamiento que contiene la aplicación. Al desplegar el archivo EAR que contiene el archivo JAR EJB del enterprise bean, también implementa el bean enterprise para el servidor GlassFish.

También puede desplegar un JAR EJB que no está contenido en un archivo EAR. La Figura 22-2 muestra el contenido de un archivo JAR EJB.



## Packaging Enterprise Beans inWAR Modules

Beans Enterprise proporcionan a menudo la lógica de negocio de una aplicación web. En estos casos, el envasado del enterprise bean dentro del módulo WAR de la aplicación web simplifica la organización y despliegue de aplicaciones. Beans Enterprise pueden ser empaquetados dentro de un módulo WAR como archivos de clase de lenguaje de programación Java o dentro de un archivo JAR que se incluye dentro del módulo WAR.  
  
To include enterprise bean class files in a WAR module, the class files should be in the

WEB-INF/classes directory.

To include a JAR file that contains enterprise beans in a WAR module, add the JAR to the WEB-INF/lib directory of the WAR module.

WAR modules that contain enterprise beans do not require an ejb-jar.xml deployment

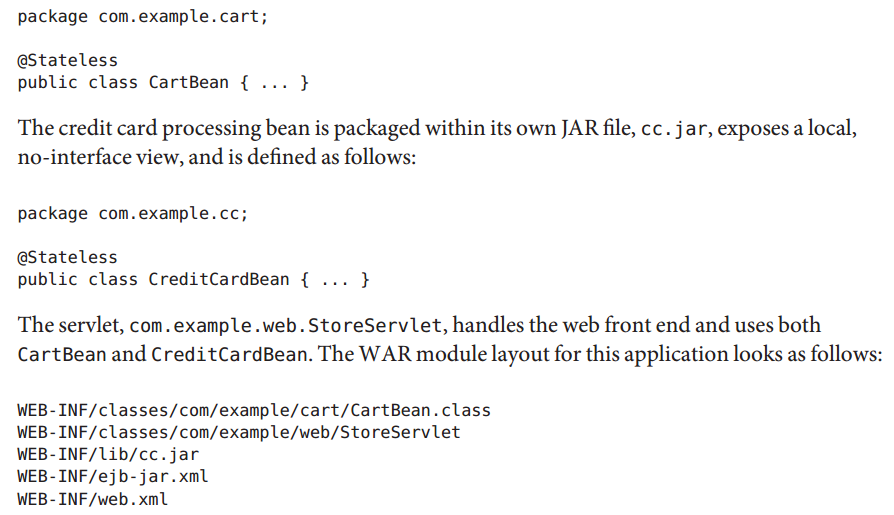
descriptor. If the application uses ejb-jar.xml, it must be located in the WAR module’s

WEB-INF directory.

Archivos JAR que contienen clases bean empresariales estándar dentro de un módulo WAR no se consideran archivos JAR EJB, incluso si el archivo JAR incluido cumple con el formato de un archivo JAR EJB.

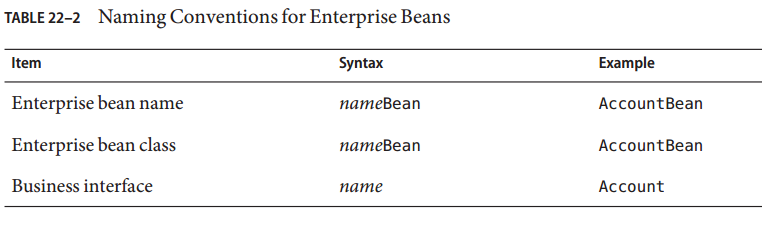
Los beans de empresa contenidos en el archivo JAR son semánticamente equivalentes a beans empresariales ubicados en el directorio del módulo WAR WEB-INF/classes, y el espacio de nombres ambiente de todos los beans de empresa están en el ámbito al módulo WAR.

Por ejemplo, supongamos que una aplicación web se compone de una empresa de carrito de la compra de frijol, una tarjeta de crédito de la empresa de procesamiento de frijol, y un extremo delantero de servlets Java. El carrito de la compra de frijol expone una, sin interfaz vista local y se define de la siguiente manera:



# Naming Conventions for Enterprise Beans

Debido a enterprise beans se componen de varias piezas, es útil seguir una convención de nomenclatura para sus aplicaciones. Tabla 22-2 resume las convenciones de los ejemplos de frijoles en este tutorial.



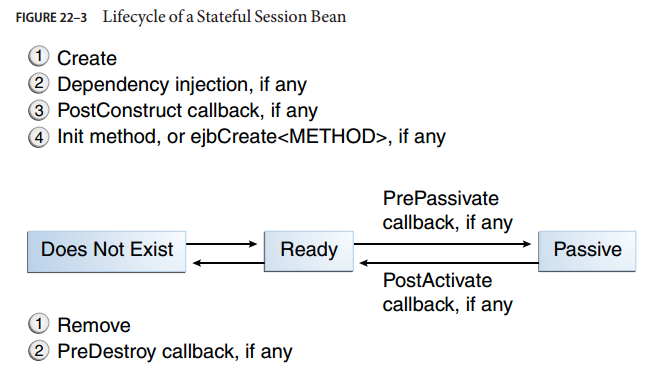
# Los ciclos de vida del Enterprise Beans

Un bean de empresa pasa por diferentes etapas durante su vida útil, o ciclo de vida. Cada tipo de grano de la empresa (con estado de sesión de sesión sin estado, sesión único, o mensaje impulsadas) tiene un ciclo de vida diferente.

Las descripciones que siguen se refieren a los métodos que se explican junto con los ejemplos de código en los dos capítulos siguientes. Si eres nuevo en los beans enterprise, usted debe saltar esta sección y ejecutar los ejemplos de código primero.

## The Lifecycle of a Stateful Session Bean

La figura 22-3 ilustra las etapas que un bean de sesión pasa a través durante su vida útil. El cliente inicia el ciclo de vida mediante la obtención de una referencia a un bean de sesión con estado. El contenedor realiza cualquier inyección de dependencias y luego llama al método anotado @PostConstruct, en su caso. El grano ya está listo para tener sus métodos de negocio invocados por el cliente.



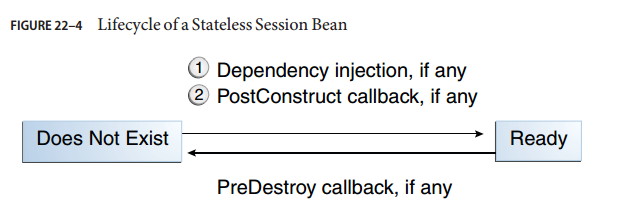
Mientras que en el escenario listo, el contenedor EJB puede decidir desactivar o pasivar, el frijol moviéndolo de memoria para el almacenamiento secundario. (Normalmente, el contenedor EJB utiliza un algoritmo de menos utilizado recientemente-para seleccionar un bean para la pasivación.) El contenedor EJB invoca el método @PrePassivate, en su caso, inmediatamente antes de la pasivación de ella. Si un cliente llama a un método de negocio en el grano mientras está en la etapa pasiva, el contenedor EJB activa el frijol, llama al método anotadoPostActivate, en su caso, y luego pasa a la etapa de listo.

Al final del ciclo de vida, el cliente invoca un método @Remove, y el contenedor EJB llama al método @PreDestroy, si los hubiere. Instancia del bean está listo para la recolección de basura.

Su código controla la invocación de un solo método de ciclo de vida: el método @Remove. Todos los otros métodos en la Figura 22-3 se invocan por el contenedor EJB.

## The Lifecycle of a Stateless Session Bean

Debido a que un bean de sesión sin estado nunca se pasiva, su ciclo de vida sólo tiene dos etapas: inexistentes y listos para la invocación de los métodos comerciales. La figura 22-4 ilustra las etapas de un bean de sesión sin estado.

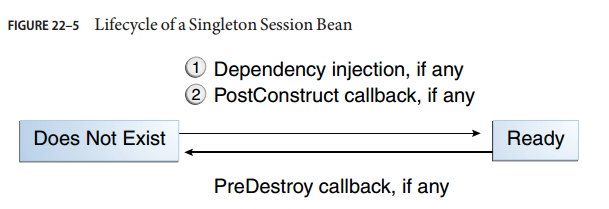


El contenedor EJB normalmente crea y mantiene un grupo de beans de sesión sin estado, a partir del ciclo de vida del bean de sesión sin estado. El contenedor realiza cualquier inyección de dependencias y luego invoca el método @PostConstruct, si existe. El grano ya está listo para tener sus métodos de negocio invocados por un cliente.

Al final del ciclo de vida, el contenedor EJB llama al método @PreDestroy, si existe. Instancia del bean está listo para la recolección de basura.

## The Lifecycle of a Singleton Session Bean

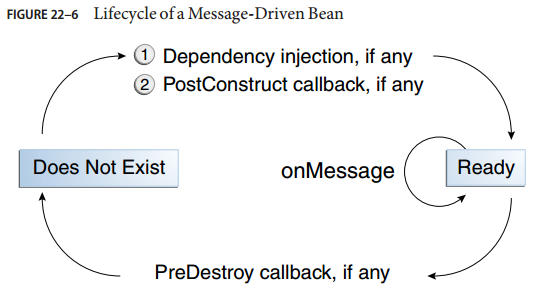
Al igual que un bean de sesión sin estado, beans de sesión singleton nunca se pasiva y tiene sólo dos etapas, inexistentes y listos para la invocación de los métodos comerciales, como se muestra en la Figura 22-5.



El contenedor EJB inicia el ciclo de vida del bean de sesión singleton mediante la creación de la instancia singleton. Esto se produce tras la implementación de la aplicación si el singleton se anota con la @Startup. El contenedor realiza cualquier inyección de dependencias y luego invoca el método @PostConstruct, si existe. El bean de sesión singleton ya está listo para tener sus métodos de negocio invocados por el cliente.

Al final del ciclo de vida, el contenedor EJB llama al método @PreDestroy, si existe. El bean de sesión singleton ya está lista para la recolección de basura.

## The Lifecycle of a Message-Driven Bean



El contenedor EJB suele crear un pool de instancias del bean controlados por mensajes. Para cada ejemplo, el contenedor EJB realiza estas tareas.

1. Si el bean controlado por mensajes utiliza la inyección de dependencia, el contenedor inyecta estas referencias antes de crear instancias de la instancia.

2. El contenedor llama al método d @PostConstruct, si los hubiere.

Al igual que un bean de sesión sin estado, un bean controlado por mensajes nunca se pasiva y tiene sólo dos estados: inexistentes y listos para recibir mensajes.

Al final del ciclo de vida, el contenedor llama al método anotadoPreDestroy, si los hubiere. Instancia del bean está listo para la recolección de basura.