Procesos de la Ingeniería Software
Tema 4

Soporte Java para construcción de aplicaciones empresariales

2. Capa de negocio en Java EE: Componentes EJB

Bibliografía

- Lectura obligada
  - Antonio Goncalvez (2013): Beginning Java EE 7, Apress
    - Capítulo 7

- Lectura complementaria
  - Erik Jendrock et al. (2014): The Java EE 7 Tutorial
    - Capítulos 32 – 34

Java EE Business Tier: Enterprise Java Beans

- Los Componentes Java EE que forman la capa de negocio de aplicaciones Java EE son los **Enterprise JavaBeans** (**EJB**)

- Definidos en la Enterprise Java Beans Specification
  - Desde la versión 3.0, basada en POJOs con anotaciones
  - Versión actual: 3.2

- Los EJB son gestionados por el contenedor de EJBs (EJB Container)
  - El contenedor gestiona aspectos como seguridad, transacciones, comunicación (RMI), etc.
  - El código del EJB se encarga exclusivamente de implementar la lógica de negocio

- Los EJB se distribuyen como:
  - Módulos Java EE desplegables independientemente (archivo JAR)
  - Módulos Java EE dentro de una aplicación empresarial Java EE (archivo EAR)

EJB es un modelo de componentes

The Enterprise JavaBeans architecture is an architecture for the development and deployment of component-based business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification
(EJB 3.1 Specification, pg 29)

Enterprise JavaBeans is a standard server-side **component model** for distributed business applications

Especificación EJB + servidor de aplicación que la implemente = Tecnología de componentes EJB

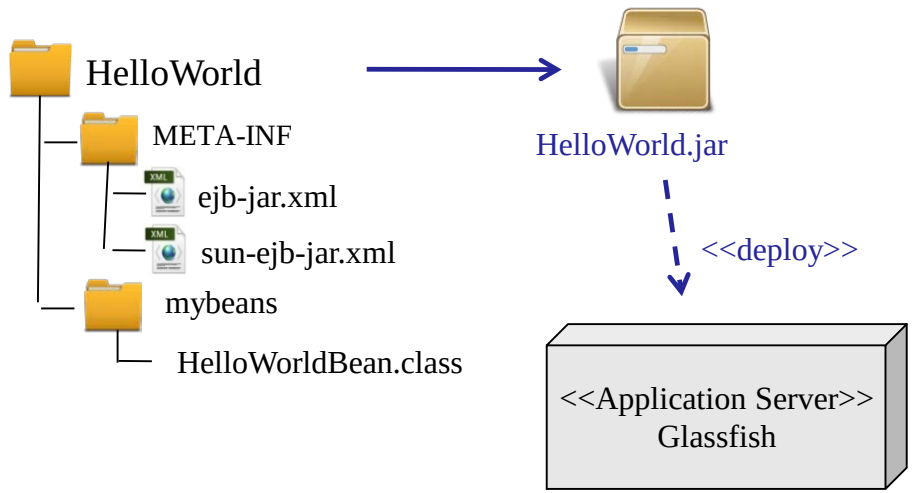
De clases POJO a EJBs

- Un EJB es un POJO con anotaciones
  - Reusable, por tanto, en otros entornos (en una aplicación Java SE, por ejemplo)

Clase POJO

```
package mybeans;
import javax.ejb.Stateless;

@Stateless
public class HelloWorldBean
{
   public String helloWorld(String name) {
           return "Hello "+name;
   }
}
```

- Un POJO se convierte en un EJB cuando es:
  1. Empaquetado en un archivo .jar
  2. Desplegado en un servidor de aplicaciones Java EE
  3. Accedido a través de un contenedor de EJBs



EJBs become such only in the context of the EJB container

Modelo de componentes EJB: Tipos de EJB

Session Beans

- Encapsulan lógica de negocio invocada de forma **síncrona** por un cliente
- Pueden implementar una o varias **interfaces de negocio**
  - Un cliente sólo puede invocar los métodos expuestos a través de la(s) interfaz de negocio
- La interfaz de negocio se puede implementar de forma:
  - Explícita
  - Implícita (no-interface view)
- Los clientes acceden al bean a través de una **referencia remota** (modelo proxy)
  - El contenedor se sitúa entre el proxy y el EJB
- Tres tipos: Stateless, Stateful y Singleton

Message-Driven Beans

- Encapsulan lógica de negocio invocada de forma **asíncrona** entre cliente y servidor
- Implementan un patrón de tipo **Listener** de **eventos**
  - En realidad es un Listener de mensajes procedentes de JMS (Java Message Service)
  - Los clientes no invocan el bean, sólo envían mensajes JMS

Modelo de componentes EJB: Interfaz de negocio

- Una interfaz de negocio debe anotarse con información acerca del tipo de acceso que va a proporcionar
  - **Acceso local** (`@Local`)
    - Acceso sólo permitido a clientes que ejecuten en la misma aplicación Java EE (i.e., en la misma JVM)
    - Los clientes pueden ser componentes web Java EE u otros beans locales
    - Valor por defecto
  - **Acceso remoto** (`@Remote`)
    - Acceso permitido a clientes que ejecutan en distintas JVM
    - Los clientes pueden ser componentes web Java EE, otros beans o aplicaciones cliente
- Reglas de uso:
  - Una misma interfaz no puede ser a la vez local y remota
  - Un bean con interfaz implícita (no-interface view) sólo puede ser accedido en modo local

Interfaz local

```
import javax.ejb.Local;

@Local
public interface CalculadoraLocal {

   public int suma(int op1, int op2);
   public int resta(int op1, int op2);

}
```

Interfaz remota

```
import javax.ejb.Remote;

@Remote
public interface CalculadoraRemote {

   public int suma(int op1, int op2);
   public int resta(int op1, int op2);

}
```

Modelo de componentes EJB: Session Beans

- **Stateless** (`@Stateless`)
  - No mantiene el estado entre invocaciones
  - Una instancia del bean por cada invocación
    - El container gestiona el pool de instancias que atienden peticiones
  - Ej: Calculadora

```
import javax.ejb.Stateless;

@Stateless
public class MyStatelessBean {
   …
}
```

- **Stateful** (`@Stateful`)
  - Mantiene el estado entre diferentes invocaciones del EJB
    - Para una misma sesión cliente/EJB
    - El estado se suele denominar Conversational State
  - La interacción se realiza con la misma instancia de EJB
  - Ej: Carrito de compra

```
import javax.ejb.Stateful;

@Stateful
public class MyStatefulBean {
   …
}
```

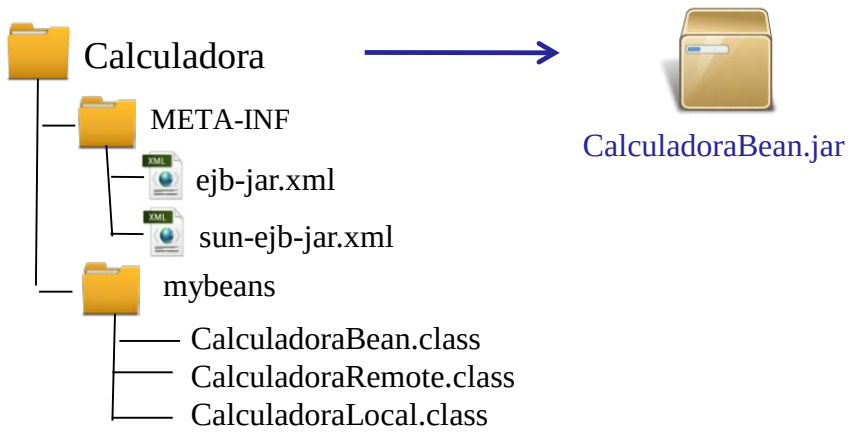
- **Singleton** (`@Singleton`)
  - Una instancia del bean única para toda la aplicación
  - Existe durante toda la vida de la aplicación (desde que es desplegado)

```
import javax.ejb.Singleton;

@Singleton
public class MySingletonBean {
   …
}
```

Desarrollo de un SessionBean

1. Definición de interfaces de negocio
   - @Local / @Remote
   - Mismas restricciones de tipos que RMI
2. Implementación del bean
   - Clase POJO
   - @Stateless/@Stateful/@Singleton
3. Empaquetamiento
   - Archivo .jar que incluya:
     - Archivos .class
     - Descriptores de despliegue necesarios
4. Despliegue en el contenedor
   - El contenedor le asocia un nombre que será utilizado para posteriores búsquedas

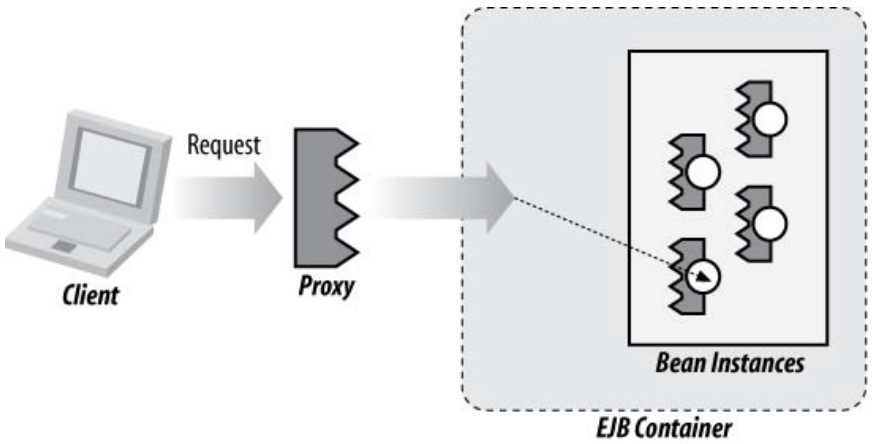
```
package mybeans;
import javax.ejb.Stateless;

@Stateless
public class CalculadoraBean implements CalculadoraLocal,
                                        CalculadoraRemote
{
   @Override
   public int suma(int op1, int op2) {
            return op1+op2;
   }

   @Override
   public int resta(int op1, int op2) {
            return op1+op2;
   }

}
```



Acceso a un SessionBean (ya desplegado)

- ¿Quién puede ser cliente de un EJB?
  - Otros EJB
  - Componentes Web Java EE: JSF, JSP o Servlets
  - Clases Java tradicionales: POJO, GUIs, etc.
  - Servicios web

- Para acceder a un SessionBean, los clientes deben obtener una referencia remota (proxy) a una instancia del Bean
  - ¡¡¡ Nunca se crea un Bean con new !!!
  - La referencia remota "pasa" siempre por el contenedor del bean



Obtención de referencia remota a un EJB

- Por **inyección de dependencias**
  - Anotación `@EJB`
  - El contenedor inyecta la referencia necesaria cuando el componente cliente es desplegado
  - Válido sólo para clientes gestionados por un servidor de aplicaciones Java EE, es decir, por un contenedor (managed components):
    - Componentes Web Java EE
    - Otros EJBs
    - Servicios Web
    - Aplicaciones Java SE ejecutadas en un Application Client Container

```
import javax.ejb.EJB;

@Stateless
public class CalculadoraClientInyeccion {

   @EJB
   private CalculadoraRemote miCalculadora;

   public static void main(String args[]) {

      miCalculadora.suma(1,2);
   }
}
```

- Por **búsquedas a través de JNDI** (JNDI lookups)
  - Se buscan los objetos a través de un servicio de nombres (directory naming service)
  - Cualquier aplicación Java (Java EE o Java SE) puede usar este mecanismo

```
import javax.naming.InitialContext;

public class CalculadoraClientJNDI {

   private CalculadoraRemote miCalculadora;

   public static void main(String args[]) {

      InitialContext ic = new InitialContext();

      miCalculadora = (CalculadoraRemote) ic.lookup
            ("java:global/calculadoraBean/
                CalculadoraBean!mybeans.CalculadoraRemote");

      miCalculadora.suma(1,2);
   }
}
```

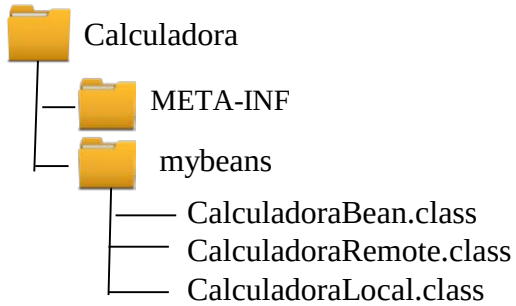
Funcionamiento básico de búsquedas JNDI

- Cuando se despliega un EJB en un contenedor, automáticamente se le asigna un identificador
  - El identificador se puede “forzar” a través de anotaciones o del descriptor ejb-jar.xml
  - Si no lo está, se le asigna uno por defecto

- Desde Java EE 6, la sintaxis es común a todos los servidores de aplicación

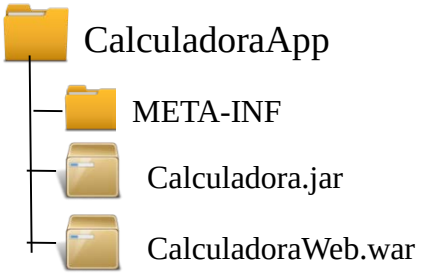
```
java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<qualified-interface-name>]
```

Calculadora.jar



java:global/Calculadora/CalculadoraBean!mybeans.CalculadoraRemote

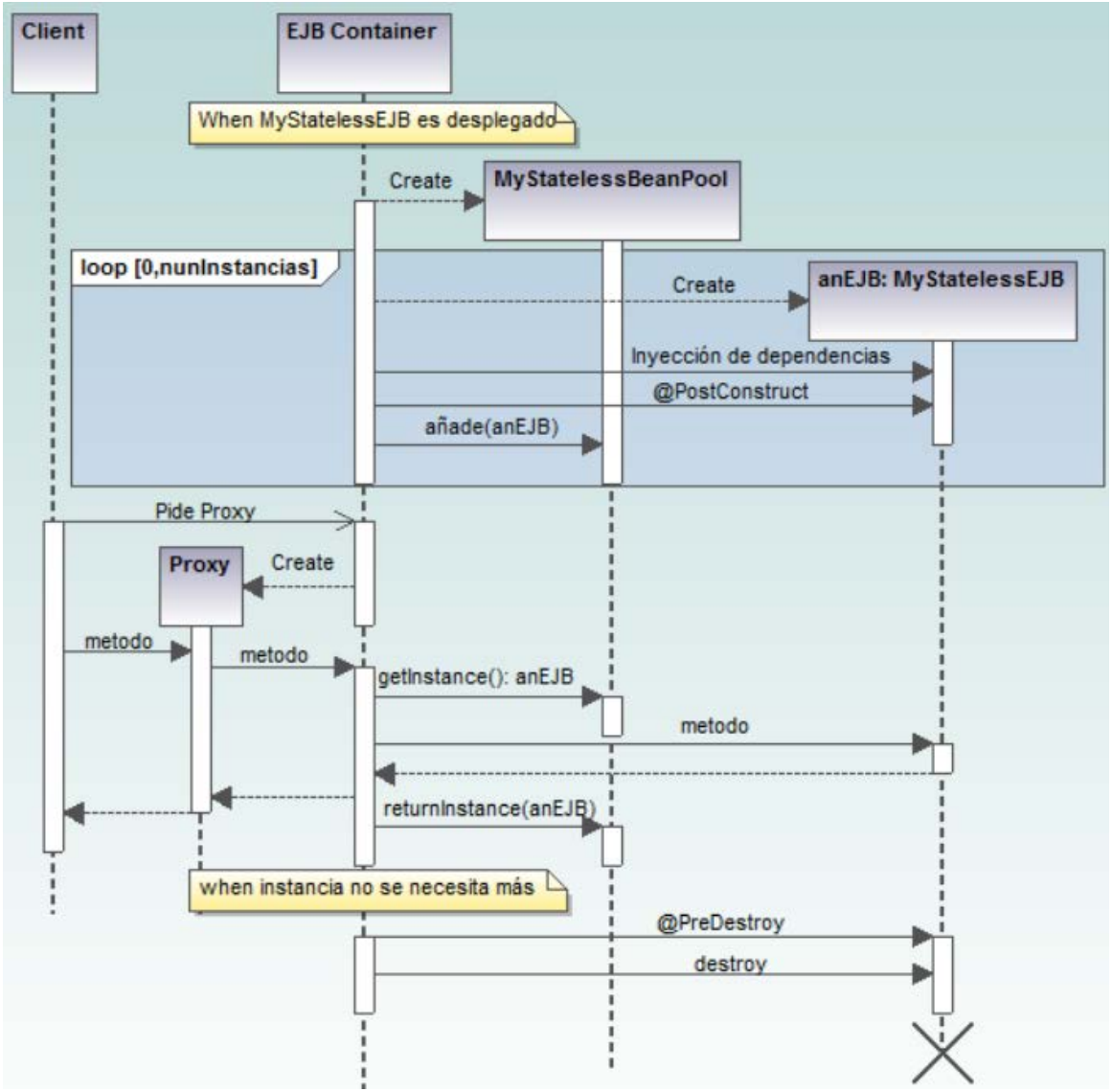
CalculadoraApp.ear



java:global/CalculadoraApp/Calculadora/CalculadoraBean!mybeans.CalculadoraRemote

- Para poder usar JNDI remotamente se requiere configurar la JVM cliente con los puertos de acceso al servicio JNDI del servidor de aplicaciones remoto
  - Dirección y puerto de escucha del contenedor (Veremos un ejemplo en ejercicios)

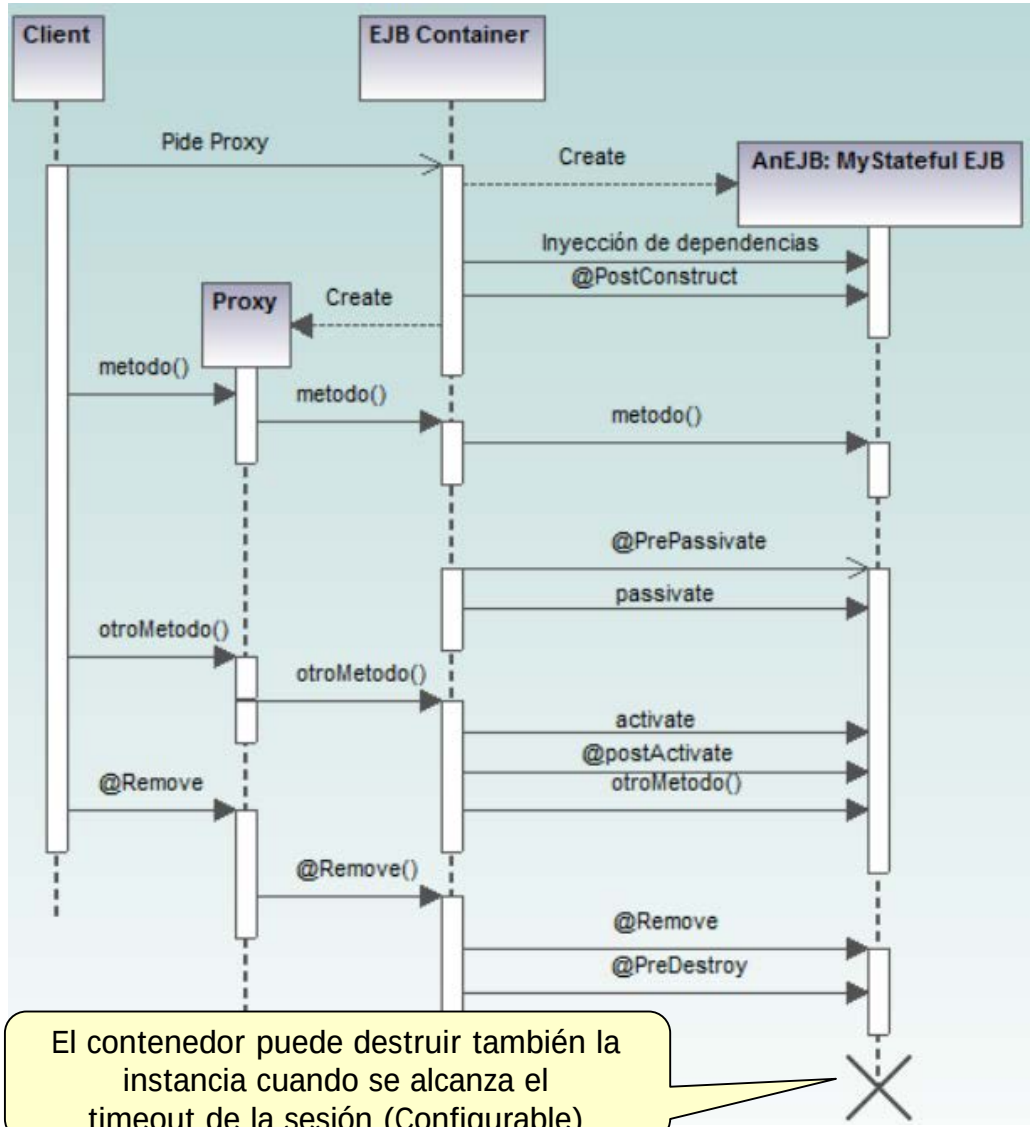
Ciclo de vida de un Stateless Bean: Callbacks




```java
import javax.ejb.*;

@Stateless
public class MyStatelessBean {

  @PostConstruct
  public void init() {
     // Some initialization activities
     // e.g. open a database connection
   }

  @PreDestroy
  public void close() {
     // Some finishing activities
     // e.g. close the database connection
   }
}
```

- Un método de tipo callback:
  - Debe retornar void
  - No puede tener parámetros
  - No puede ser ni static ni final
- Todos los callbacks son opcionales
- El ciclo de vida de un Singleton es similar, pero con una única instancia

Ciclo de vida de un StatefulBean: Callbacks




```java
import javax.ejb.*;

@Stateful
public class MyStatefulBean {

  @PostConstruct
  public void init() {
     // Some initialization activities
     // e.g. open a database connection
   }

  @PreDestroy
  public void close() {
     // Some finishing activities
     // e.g. close the database connection
   }

  @PrePassivate
  public void closeDatabaseConnection() {
     // e.g. close the database connection
   }

  @PostActivate
  public void reopenDatabaseConnection() {
     // e.g. close the database connection
   }

  @Remove
  public void borraInstancia() {}
}
```

Anotaciones EJB: SessionContext

- El **SessionContext** representa el enlace entre la instancia del EJB y el contenedor

- A través de la referencia al SessionContext se puede:
  - Conocer información de contexto de la invocación concreta
    - El cliente que realiza la invocación o su ubicación
  - Acceder de manera explícita a los servicios del contenedor
    - La mayoría de ellos se soportan de manera transparente (inyección, comunicación remota, transacciones, mantenimiento del estado, etc.) pero en ocasiones puede ser necesario acceder de este modo

- La referencia al contexto desde un SessionBean se obtiene también mediante inyección de dependencias

```
@Stateless
public class CalculadoraBean implements CalculadoraRemote {

   @Resource
   private SessionContext context;

}
```

Descriptor de despliegue EJB: ejb-jar.xml

- El descriptor de despliegue (ejb-jar.xml) se puede usar:
  - Como medio alternativo a las anotaciones para configurar el EJB
    - Más flexible (los EJB son clases POJO puras) pero algo más complejo
  - Para sobrescribir la configuración de las anotaciones
  - Para configurar aspectos no abordados por las anotaciones
    - Por ejemplo, asignar propiedades de configuración del bean que dependen del entorno de despliegue

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="3.1" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd">
<enterprise-beans>
      <session>
         <ejb-name>CalculadoraBean</ejb-name>
         <ejb-class>mybeans.CalculadoraBean</ejb-class>
         <local>mybeans.CalculadoraLocal</local>
         <remote>mybeans.CalculadoraRemote</remote>
         <session-type>Stateless</session-type>
      </session>
   </enterprise-beans>
</ejb-jar>
```

Descriptor equivalente a las anotaciones vistas en la trasparencia 4.25
(Si lo usamos, el código no tendría ninguna referencia a EJB)

Descriptor de despliegue: Variables de entorno

- Un uso común del descriptor de despliegue es dar valor a las denominadas variables de entorno (Environment Entries)
  - Variables que dependen del entorno de despliegue del EJB
  - Basado en inyección de dependencias

```
@Stateless
public class ItemEJB {

   @Resource(name = "currencyEntry")
   private String currency;

   @Resource(name = "changeRateEntry")
   private Float changeRate;

   public Item convertPrice(Item item) {
      item.setPrice(item.getPrice() * changeRate);
      item.setCurrency(currency);
      return item;
   }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="3.1"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd">
   <enterprise-beans>
      <session>
         <ejb-name>ItemEJB</ejb-name>
         <ejb-class>ItemEJB</ejb-class>
         <env-entry>
            <env-entry-name>currencyEntry</env-entry-name>
            <env-entry-type>java.lang.String</env-entry-type>
            <env-entry-value>Euros</env-entry-value>
          </env-entry>
          <env-entry>
            <env-entry-name>changeRateEntry</env-entry-name>
            <env-entry-type>java.lang.Float</env-entry-type>
            <env-entry-value>0.80</env-entry-value>
         </env-entry>
       </session>
   </enterprise-beans>
</ejb-jar>
```

Modelo de componentes EJB: Message-Driven Beans

- **Message-Driven Beans** (@MessageDriven)
  - Es un bean que permite implementar interacciones asíncronas
    - No invocaciones a través de una interfaz
  - Actúa como Listener de mensajes JMS (Java Message Service)
    - Ofrece un método único onMessage que es invocado por el contenedor cuando se recibe un mensaje
  - Los mensajes pueden ser enviados por:
    - Componentes Java EE: Otros EJB, aplicaciones cliente o componentes web
    - Aplicaciones no Java EE que soporten JMS
  - Se podría ver como un Stateless Session Bean asíncrono
    - Sin estado
    - El contenedor maneja un pool de instancias

Servicios del contenedor de EJBs

- Acceso remoto vía RMI o IIOP
- Inyección de dependencias
  - Otros EJBs, fuentes de datos (DataSource), ServiceContext, variables de entorno, etc.
- Servicio de nombres (JNDI)
- Gestión del ciclo de vida (callbacks)
- Concurrencia
  - Todos los EJB son thread-safe por naturaleza, en los Singleton se puede configurar
- Gestión del estado (Stateful Beans)
- Pools de instancias (Stateless Beans)
- Transacciones
- Seguridad (Ver Tema 4.5)
- Timers
  - Lanzamiento de actividades temporizadas
- Interceptores

Transacciones en EJBs

Container-managed transactions

- Transacciones gestionadas por el contenedor en base a anotaciones o descriptor de despliegue
- Anotación **@TransactionAttribute**:
  - `TransactionAttributeType.REQUIRED`
    - El método del EJB se ejecuta en la transacción del cliente si existe, o sino se crea una para él
    - Valor por defecto => Todo método de un EJB se ejecuta siempre en una transacción
  - `TransactionAttributeType.REQUIRES_NEW`
    - Se crea una nueva transaccción propia para el método del EJB. La transacción del cliente se suspende
  - `TransactionAttributeType.SUPPORTS`
    - El método del EJB se ejecuta en la transacción del cliente si existe, o sino en ninguna
  - `TransactionAttributeType.MANDATORY`
    - El método del EJB se ejecuta en la transacción del cliente.
    - Si no existe se lanza una excepción
  - `TransactionAttributeType.NOT_SUPPORTED`
    - El método del EJB no se ejecuta dentro de ninguna transacción (si el cliente tiene, se suspende)
  - `TransactionAttributeType.NEVER`
    - El método del EJB no se ejecuta dentro de ninguna transacción (si el cliente tiene, se lanza una excepción)

Bean-managed transactions

- Transacciones gestionadas desde los propios beans de manera programática
  - JTA API

CMT vs BMT

CMT

```
@Stateless
public class BooksManagementeEJB {

@EJB
private BooksDAO books;

public Book bookReturned(long id) {
   Book b = books.getBook(id);
   b.setAvailable();
   books.updateBook(b);
}
```

BMT

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class BooksManagementeEJB {

@EJB
private BooksDAO books;

@Resource
private UserTransaction ut;

public Book bookReturned(long id) {
   try {
      ut.begin()

      Book b = books.getBook(id);
      b.setAvailable();
      books.updateBook(b);

      ut.commit();

   } catch (Exception e) {
      ut.rollback();
   }
}
```

SessionSynchronization

- La interfaz **SessionSynchronization** define un conjunto de callbacks que un EJB puede implementar y que serán invocados por el container durante la gestión de transacciones:
  - afterBegin()
  - beforeCompletion()
  - afterCompletion()

- Útil en caso de Stateful Beans, que mantienen la transacción a través de diferentes invocaciones

- Un EJB que usa CMT puede indicar el Rollback de una transacción, por ejemplo, cuando se produce una excepción, usando el SessionContext:

```
sessionContext.setRollbackOnly()
```

- Ver ejemplo en enlace disponible en Moodle

Manejo de excepciones en SessionBeans

- Dos tipos de excepciones en aplicaciones Java EE
  - **System Exception**
    - Son las generadas por los servicios subyacentes: no funciona una inyección de dependencias, no se puede establecer conexión con la base de datos, etc.
    - Extienden a RuntimeException
    - El contenedor mapea todas las excepciones que extienden a RuntimeException a una EJBException => Errores inesperados, el cliente no es responsable de manejarlas
    - Provocan el “rollback” de la transacción
  - **Application Exception**
    - Son las generadas por la propia lógica de la aplicación
    - Deben extender a Exception
    - El contenedor las envía al cliente como tal => El cliente es responsable de manejarlas
    - No provocan el “rollback” de la transacción (responsabilidad del cliente)
    - Si queremos que lo provoquen, se pueden utilizar la siguiente anotación:

      @ApplicationException (rollback=true)