Procesos de la Ingeniería Software
Tema 4

Soporte Java para construcción de aplicaciones empresariales

2. Capa de negocio en Jakarta EE: Jakarta Enterprise Beans

Bibliografía

- Básica
  - Oracle and affiliates (2021): The Jakarta EE Tutorial (Release 9.1)
    - Parte VII: Enterprise Beans (Capítulos 35 – 38)

- Complementaria
  - Antonio Goncalvez (2013): Beginning Java EE 7, Apress
    - Capítulos 7 y 8 (Apartado Session Beans Lifecycle)
  - Eclipse Foundation: Jakarta Enterprise Beans Core Features 4.0
    - https://jakarta.ee/specifications/enterprise-beans/4.0/

Jakarta EE Business Tier: Enterprise Beans

- La **capa de negocio** de aplicaciones Jakarta EE se implementa en base al desarrollo de componentes **Jakarta Enterprise Beans**
  - Definidos en la Jakarta Enterprise Beans Core Features Specification
    - Ultima versión: 4.0
    - En Java EE se denominaban Enterprise Java Beans (EJB)

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

- Los Enterprise Beans se empaquetan para despliegue como:
  - Módulos Jakarta EE desplegables independientemente (archivo **JAR**)
  - Módulos Jakarta EE (archivo jar) dentro de una aplicación empresarial (archivo **EAR**)

Enterprise Beans es un modelo de componentes

The Enterprise Beans architecture is an architecture for the development and deployment of component-based business applications.
Applications written using the Enterprise Beans 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 Beans specification (Enterprise Beans Specification)

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

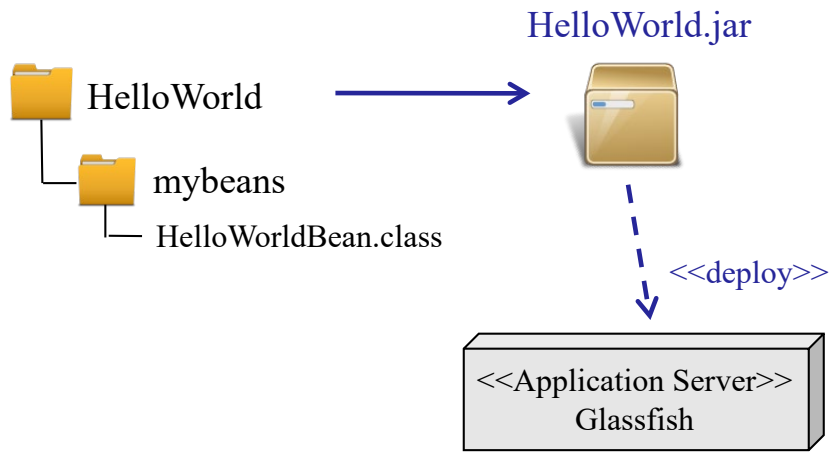
Especificación Enterprise Beans + servidor de aplicación que la implemente = Tecnología de componentes Enterprise Beans

De clases POJO a Enterprise Beans

- Un Enterprise Bean es en realidad una clase POJO con anotaciones
  - Reusable, por tanto, en otros entornos (en una aplicación Java SE, e.g.)

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

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

- Se convierte en un Enterprise Bean cuando, después de ser empaquetado en un .jar, es desplegado en un servidor de aplicación
  - Son objetos remotos accesibles mediante RMI cuya creación y gestión es responsabilidad del Enterprise Bean Container



Enterprise Beans become such only in the context of the Enterprise Bean Container

Tipos de Enterprise Beans

Session Beans

- Encapsulan lógica de negocio invocada programáticamente
  - Habitualmente de forma **síncrona** aunque puede admitir invocaciones asíncronas
- Pueden implementar una o varias **interfaces 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 bean
- Tres tipos: Stateless, Stateful y Singleton

Message-Driven Beans

- Encapsulan lógica de negocio invocada de forma **asíncrona**
- Implementan un patrón de tipo **Event Listener**
  - Son objetos Listener de mensajes procedentes de JMS (Jakarta Message Service)
  - Los clientes no invocan el bean, sólo envían mensajes al JMS

Interfaz de negocio de Enterprise Beans

- Las interfaces de negocio deben anotarse para indicar el tipo de acceso que van a proporcionar
  - **Acceso local** (`@Local`)
    - Acceso permitido solo a clientes que ejecuten en la misma JVM (mismo servidor)
    - Los clientes pueden ser componentes web Java EE u otros Enterprise Beans
    - Es el valor por defecto
  - **Acceso remoto** (`@Remote`)
    - Acceso permitido a clientes que ejecutan en distintas JVM
    - Los clientes pueden ser componentes web Java EE, otros EJB o aplicaciones de escritorio
- Reglas de uso:
  - Una misma interfaz no puede ser a la vez local y remota
  - Un Enterprise Bean con interfaz implícita (no-interface view) sólo puede ser accedido en modo local

Interfaz local

```
import jakarta.ejb.Local;

@Local
public interface ICalculadoraLocal {

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

}
```

Interfaz remota

```
import jakarta.ejb.Remote;

@Remote
public interface ICalculadoraRemote {

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

}
```

Tipo de Session Beans

- **Stateless** (`@Stateless`)
  - No mantiene estado entre invocaciones
  - Cada invocación es atendida por una instancia diferente
    - El contenedor gestiona el pool de instancias disponibles
  - Ej: Calculadora

```
import jakarta.ejb.Stateless;

@Stateless
public class MyStatelessBean {
   …
}
```

- **Stateful** (`@Stateful`)
  - Mantiene estado entre diferentes invocaciones del bean
    - Para una misma sesión cliente-bean
    - El estado se suele denominar Conversational State
  - Sesión completa atendida por la misma instancia
  - Ej: Carrito de compra

```
import jakarta.ejb.Stateful;

@Stateful
public class MyStatefulBean {
   …
}
```

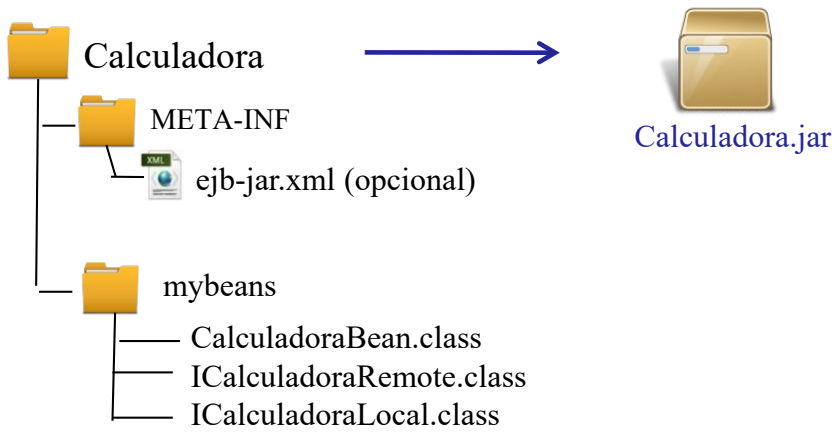
- **Singleton** (`@Singleton`)
  - Una instancia del bean única para toda la aplicación
  - Existe durante toda la vida de la aplicación (desde que es desplegado)

```
import jakarta.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 y accesos

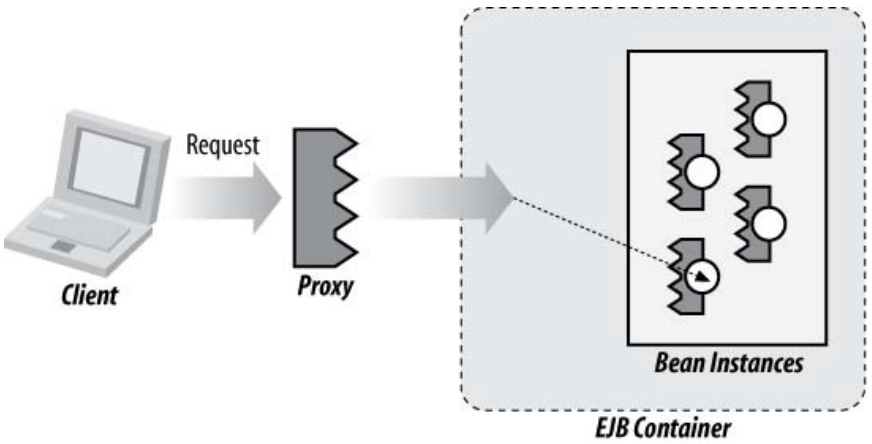
```
package mybeans;
import jakarta.ejb.Stateless;

@Stateless
public class CalculadoraBean implements ICalculadoraLocal,
                                        ICalculadoraRemote  {
   public int suma(int op1, int op2) {
            return op1+op2;
   }

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

}
```



Acceso a un SessionBean (ya desplegado)

- ¿Quién puede ser cliente de un Enterprise Bean?
  - Otros Enterprise Bean
  - Componentes Web Java EE: páginas JSF o JSP, Servlets
  - Clases Java tradicionales: POJO, clases Swing, etc.
  - Servicios web

- Los clientes deben obtener una referencia remota (proxy) a la instancia del Bean
  - ¡¡¡ Nunca se crea una instancia de Enterprise Bean con new !!!
  - La referencia remota "pasa" siempre por el contenedor del bean



Obtención de referencia remota a un SessionBean

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

```
import jakarta.ejb.EJB;

public class CalculadoraClientInyeccion {

   @EJB
   private ICalculadoraRemote 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 o componente Java (Java EE o Java SE) puede usar este mecanismo

```
import javax.naming.InitialContext;

public class CalculadoraClientJNDI {

   private ICalculadoraRemote miCalculadora;

   public static void main(String args[]) {

      InitialContext ic = new InitialContext();

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

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

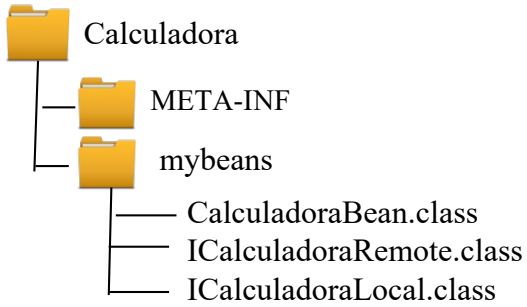
Funcionamiento básico de búsquedas JNDI

- Cuando se despliega un bean en un contenedor, automáticamente se le asigna un identificador

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

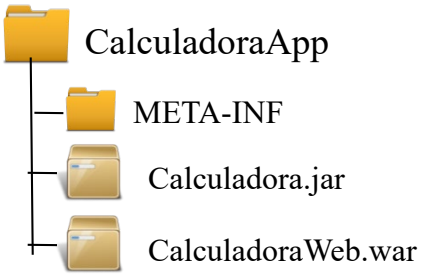
```
java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<qualified-interface-name>]
  scope = global | app | module
```

Calculadora.jar



java:global/Calculadora/
CalculadoraBean!mybeans.ICalculadoraRemote

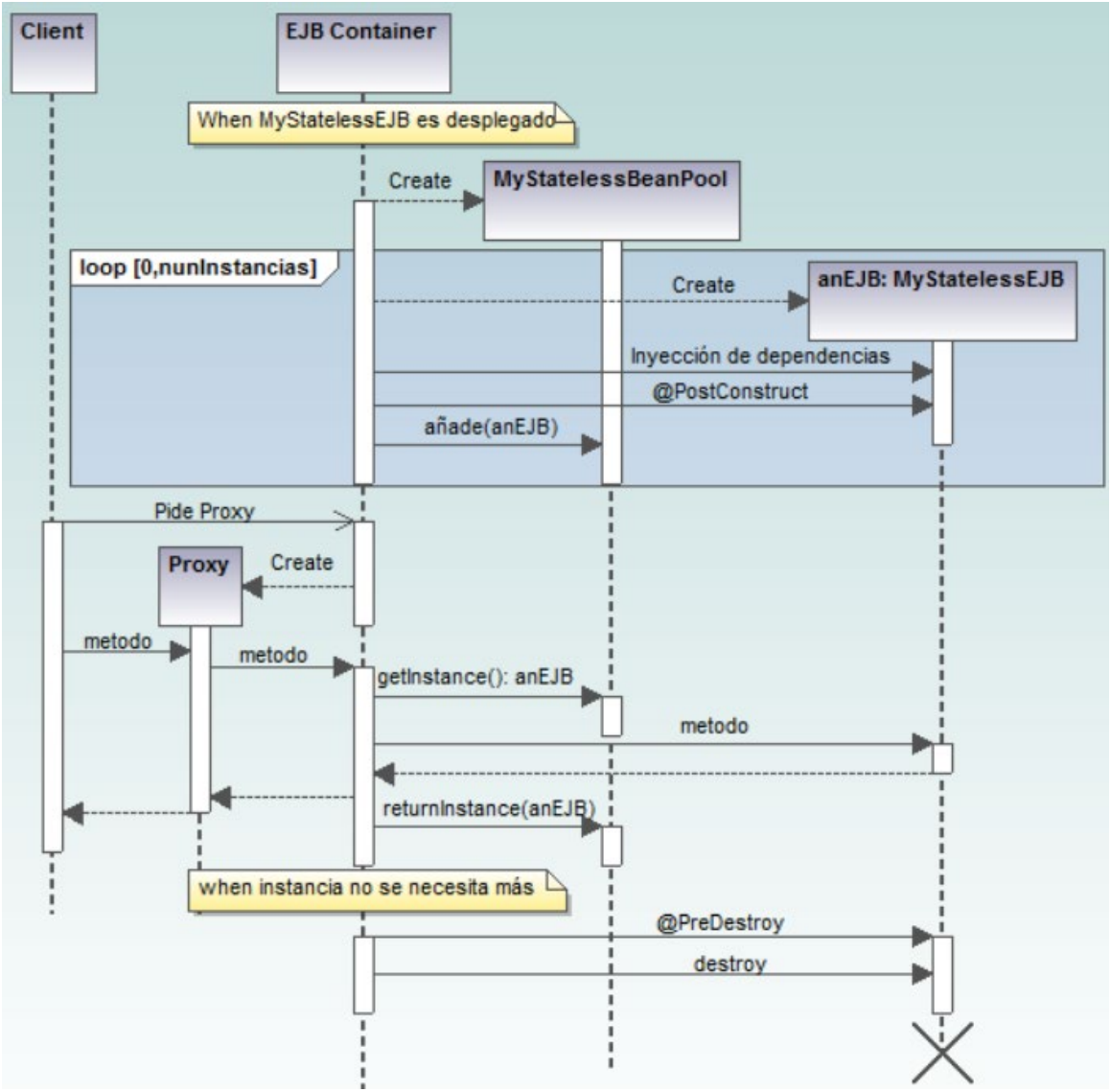
CalculadoraApp.ear



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

- 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

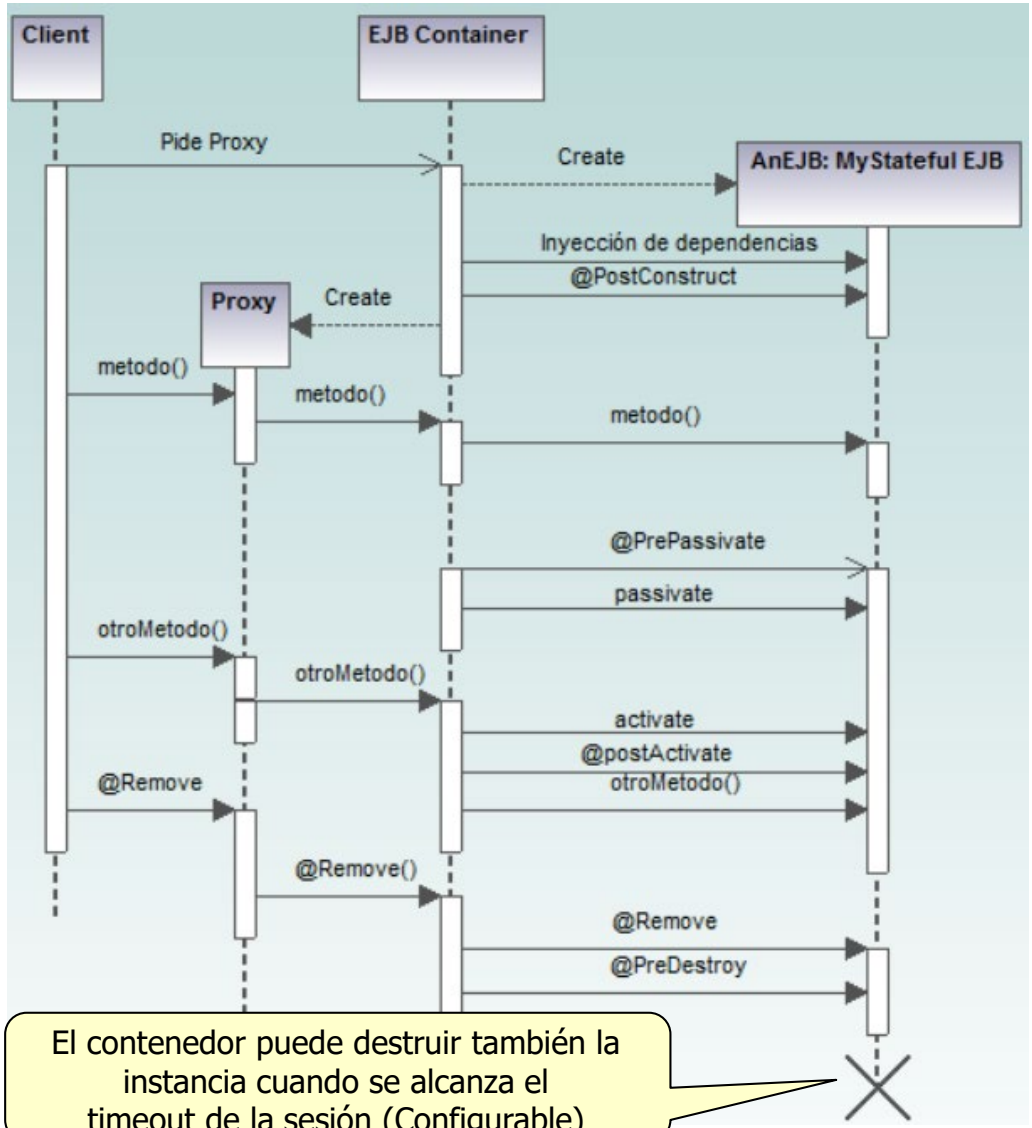
Ciclo de vida de un Stateless Bean - Callbacks




```
@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




```
@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() {}

}
```

SessionContext

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

- A través de la referencia al SessionContext se puede:
  - Conocer información de contexto de la invocación concreta
    - Por ejemplo, información sobre 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 ICalculadoraRemote {

   @Resource
   private SessionContext context;

}
```

Manejo de excepciones en SessionBeans

- Dos tipos de excepciones
  - **System Exception**
    - 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 extiendan a RuntimeException a una EJBException => Errores inesperados, el cliente no es responsable de manejarlas
    - Provocan el "rollback" de la correspondiente transacción
  - **Application Exception**
    - Generadas por la propia lógica de la aplicación
    - 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)

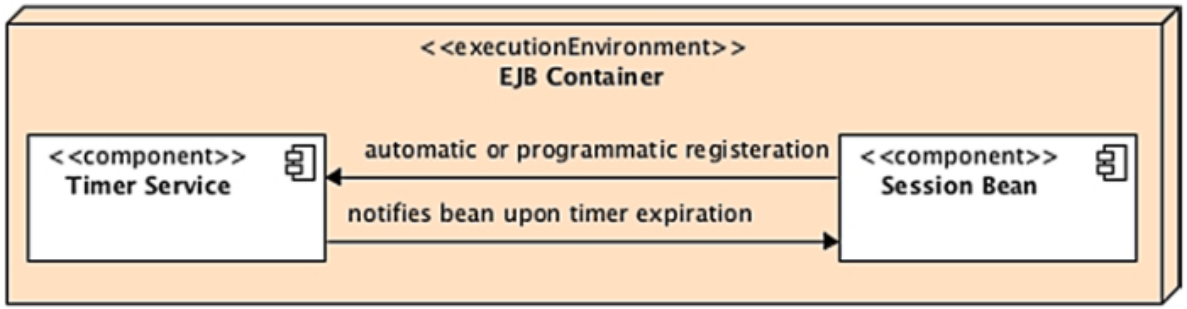
Servicios ofrecidos por el EJBContainer

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

TimerService

- Los Enterprise Beans pueden programar tareas temporizadas a través del **TimerService** proporcionado por el contenedor
  - El acceso al TimerService se consigue:
    - Por inyección de dependencias (@Resource)
    - A través del SessionContext (SessionContext.getTimerService())

- Modo de uso:
  - El Bean programa eventos temporizados programática o automáticamente
  - Cuando se alcanza el tiempo de expiración, el contenedor notifica al Bean mediante la invocación de un método de callback
  - Los timers creados son persistentes por defecto



Uso programático del TimerService

- El bean programa el evento temporizado a través de la interfaz `TimerService:`
  - Diferentes métodos para creación y gestión de eventos temporizados:
    - createSingleActionTimer: Creates a single-action timer that expires at a given point in time or after a specified duration. The container removes the timer after the timeout callback method has been successfully invoked.
    - createIntervalTimer: Creates an interval timer whose first expiration occurs at a given point in time and whose subsequent expirations occur after specified intervals.
    - getAllTimers: Return all the active timers
  - https://jakarta.ee/specifications/platform/9/apidocs/jakarta/ejb/timerservice
  - Las expresiones de tiempo se manejan usando la clase `ScheduleExpression`

- El Bean define el método de callback mediante la anotación `@Timeout`
  - Debe retornar void
  - Opcionalmente puede recibir el propio Timer como parámetro para proporcionar información necesaria para realizar la tarea asociada al evento

Uso programático del TimerService - Ejemplo

```
@Stateless
public class CustomersManagementBean {

   @Resource
   private TimerService timerService;

   @EJB
   private IMailService mailService;

   @PersistenceContext(unitName = “clientes")
   private EntityManager em;

   public void createCustomer(Customer customer) {
      // Se añade el nuevo cliente a la base de datos
      em.persist(customer);

      // Programación de un evento para felicitar al cliente por su cumpleaños
      ScheduleExpression birthDay = new ScheduleExpression().dayOfMonth(customer.getBirthDay())
                                                            .month(customer.getBirthMonth());
      timerService.createCalendarTimer(birthDay, new TimerConfig(customer, true));
   }

   @Timeout
   public void sendBirthdayEmail(Timer timer) {
      Customer customer = (Customer) timer.getInfo();
      mailService.sendEmail(customer.email(), “Happy Birthday!”);
   }
}
```

Uso automático del TimerService

- Basado en el uso de la anotación `@Schedule`
  - Identifica el método de callback
  - Y permite definir la temporización del evento

- El timer se programa automáticamente cuando se despliega el bean

```
@Singleton
public class GestionNominasBean {

   @EJB
   private IEmpleadosDAO daoEmpleados;

   @Schedule(dayOfMonth="1", month="Jan")
   public void subeSueldos() {
      List<Empleado> empleados = daoEmpleados.empleados();
      for (Empleado e: empleados) {
         e.aumentaSueldo(0,5);
      }
   }
}
```

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>
```

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