

Tema 3: Tecnologías para el desarrollo en el servidor

Servlets y Java Server Pages (JSP)

Java Persistence API (JPA)

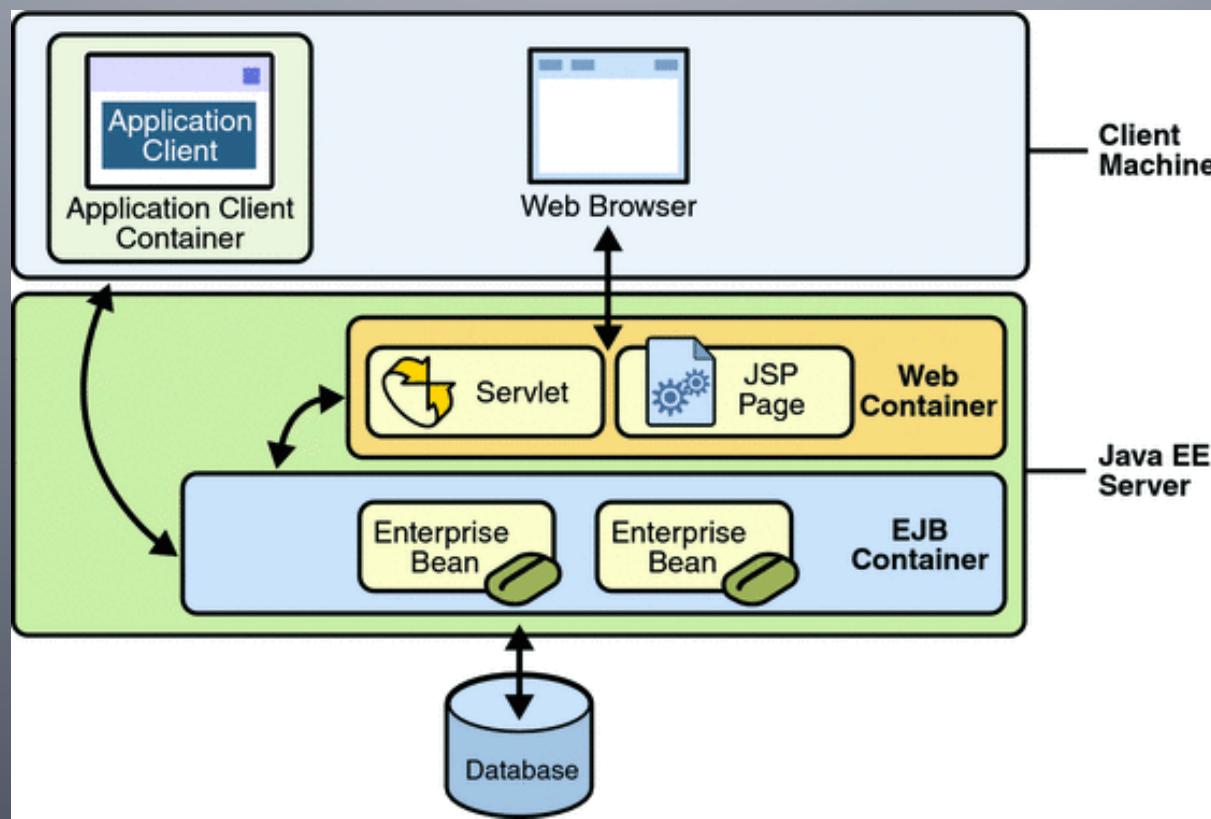
Java Server Faces (JSF)

Enterprise Java Beans (EJB)

Sistemas de Información para Internet
3º del Grado de Ingeniería Informática (tres menciones)

Departamento de Lenguajes y Ciencias de la Computación
Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Enterprise Java Bean (EJB)



Enterprise Java Beans

Motivación

- Hemos visto cómo JPA se utiliza para simplificar la persistencia de los datos en BBDD (capa de acceso a los datos)
- Por otro lado, hemos visto cómo presentar la información con ayuda de JSF (capa de presentación)
- Nos falta **conectar ambas capas**: capa de negocio
- En Java EE la capa de negocio está compuesta por **Enterprise Java Beans**
- El contenedor de EJBs será el encargado de ofrecer servicios de transacciones, concurrencia y escalabilidad

Enterprise Java Beans

Tipos

- Los dos tipos de EJBs más importantes son:
 - **Session beans** (no confundir con ámbito de sesión): son normalmente **inyectados** en los *backing bean* e implementan la lógica de negocio
 - **Message-driven beans**: utilizados para integrar la aplicación con otras aplicaciones externas
 - Basada en el envío de mensajes con la API JMS (Java Message Service).

Enterprise Java Beans

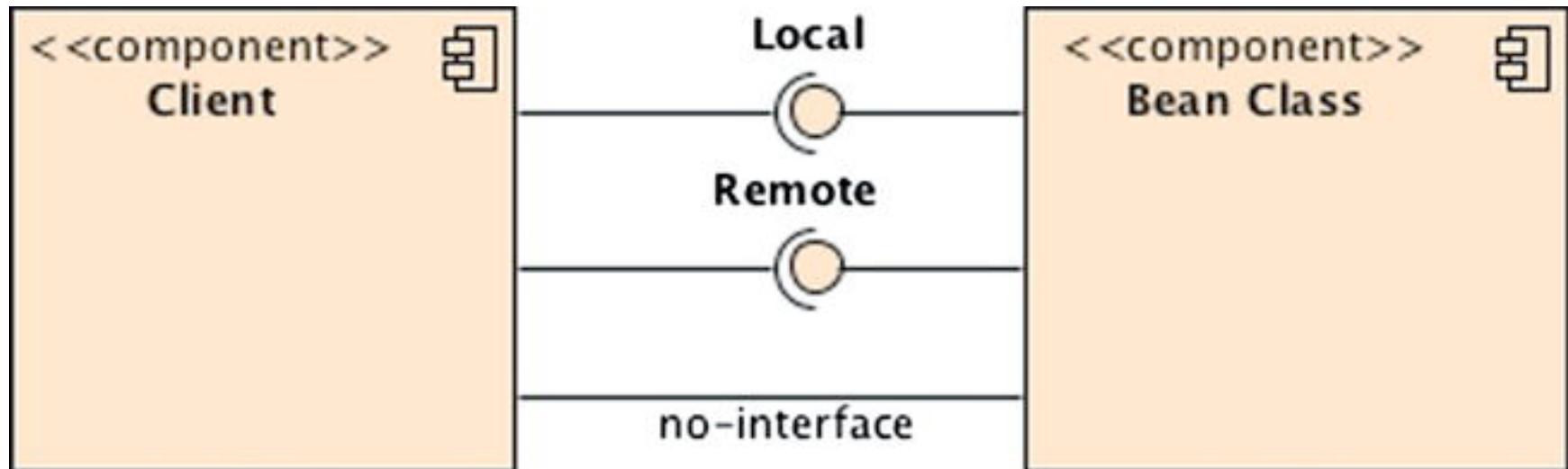
Session Beans: estructura

- Un *session bean* está formado por:
 - Un **bean** (implementación)
 - Clase pública, no abstracta ni *final*
 - Con constructor sin argumento
 - Que no defina *finalize()*
 - Los **métodos de negocio** no deben comenzar por **ejb**, no deben ser *final* ni *static*
 - Debe estar anotada con **@Stateless**, **@Stateful** o **@Singleton**

Enterprise Java Beans

Session Beans: estructura

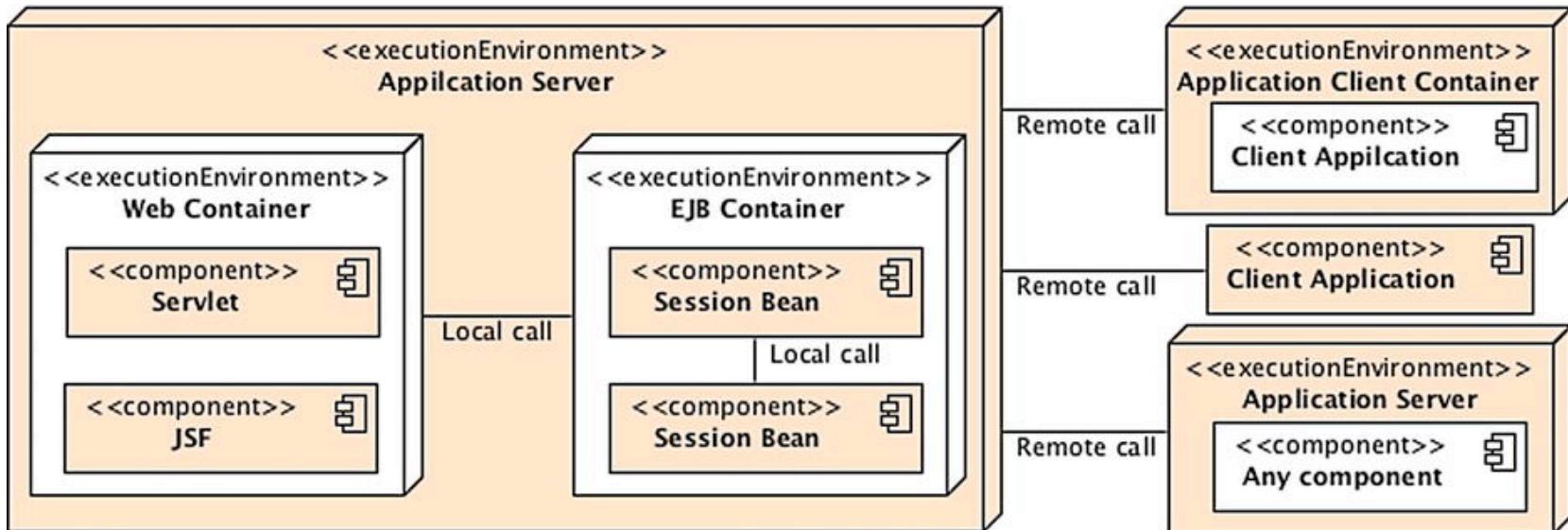
- Un *session bean* está formado por (cont.):
 - Una **interfaz**: declara los métodos a los que se puede acceder
 - Tres tipos: **local**, **remota**, **sin interfaz**



Enterprise Java Beans

Session Beans: interfaces

- ¿Cuándo usamos las distintas interfaces?
 - **Local**: cuando el cliente del EJB y el EJB se encuentran en la misma JVM (el caso de un servidor de aplicaciones)
 - **Remota**: cuando el cliente y el EJB se encuentran en JVMs diferentes



Enterprise Java Beans

Session Beans: interfaces

- ¿Qué diferencia hay entre ambas?
 - **Local**: los parámetros se pasan por referencia (excepto tipos básicos), es decir, se pasan las referencias a los objetos
 - **Remota**: los parámetros se pasan por valor, es decir, hay que serializarlos y se utiliza RMI (*Remote Method Invocation*)

Enterprise Java Beans

Session Beans: interfaces

- ¿Cómo se declaran las interfaces?
- Hay dos opciones
 - Anotando las interfaces

```
@Local  
public interface ItemLocal {  
    List<Book> findBooks();  
    List<CD> findCDs();  
}
```

```
@Remote  
public interface ItemRemote {  
    List<Book> findBooks();  
    List<CD> findCDs();  
    Book createBook(Book book);  
    CD createCD(CD cd);  
}
```

La interfaz remota en este caso declara dos métodos más

```
@Stateless
```

```
public class ItemEJB implements ItemLocal, ItemRemote {  
    // ...  
}
```

Enterprise Java Beans

Session Beans: interfaces

- ¿Cómo se declaran las interfaces?
- Hay dos opciones (cont.)
 - Anotando el EJB

```
public interface ItemLocal {  
    List<Book> findBooks();  
    List<CD> findCDs();  
}  
  
public interface ItemRemote {  
    List<Book> findBooks();  
    List<CD> findCDs();  
    Book createBook(Book book);  
    CD createCD(CD cd);  
}  
  
@Stateless  
@Remote(ItemRemote.class)  
@Local(ItemLocal.class)  
@LocalBean  
public class ItemEJB implements ItemLocal , ItemRemote {  
    // ...  
}
```

Especialmente útil si no podemos modificar la interfaz por ser código heredado

Enterprise Java Beans

Session Beans: interfaces

- Un *bean sin interfaz* es una variante de un bean con interfaz local en la que los métodos públicos son expuestos
- En cuanto un *bean* declara una interfaz pierde la “*no-interfaz*”
- Para mantenerla es necesario usar la anotación **@LocalBean**

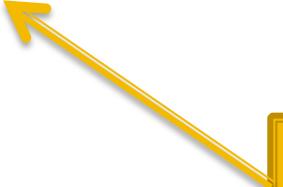
```
@Stateless  
@Remote(ItemRemote.class)  
@Local(ItemLocal.class)  
@LocalBean  
public class ItemEJB implements ItemLocal , ItemRemote {  
    // ...  
}
```

Mantiene la no-interfaz a pesar de haber declarado una remota y otra local

Enterprise Java Beans

Session Beans: tipos

- Existen tres tipos de *session beans*:
 - Sin estado (**stateless**): se declaran con la anotación `@Stateless`
 - Con estado (**stateful**): se declaran con la anotación `@Stateful`
 - Singular (**singleton**): se declaran con la anotación `@Singleton`

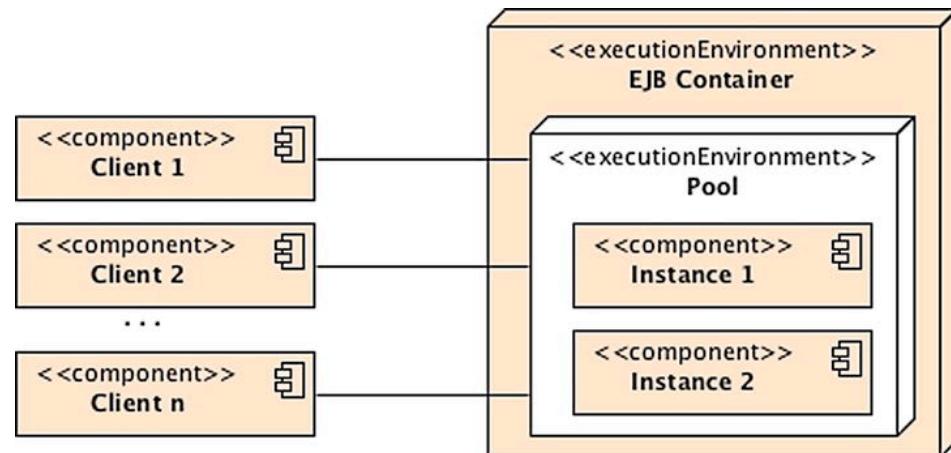


Puede ser invocado concurrentemente

Enterprise Java Beans

Session Beans: Stateless

- Los EJB sin estado no guardan ninguna información entre llamadas a sus métodos de negocio
- Esto implica que deben resolver la tarea en una única llamada a método
- Son los más eficientes, puesto que al no tener estado el servidor puede compartir las instancias entre múltiples clientes
- El contenedor EJB mantendrá un *pool* de instancias por cuestiones de eficiencia
- Cada vez que se invoca un método de un *Stateless*, el contenedor toma una instancia cualquiera del pool y luego la devuelve



Enterprise Java Beans

Session Beans: Stateless

Ejemplo

Inyección de dependencias

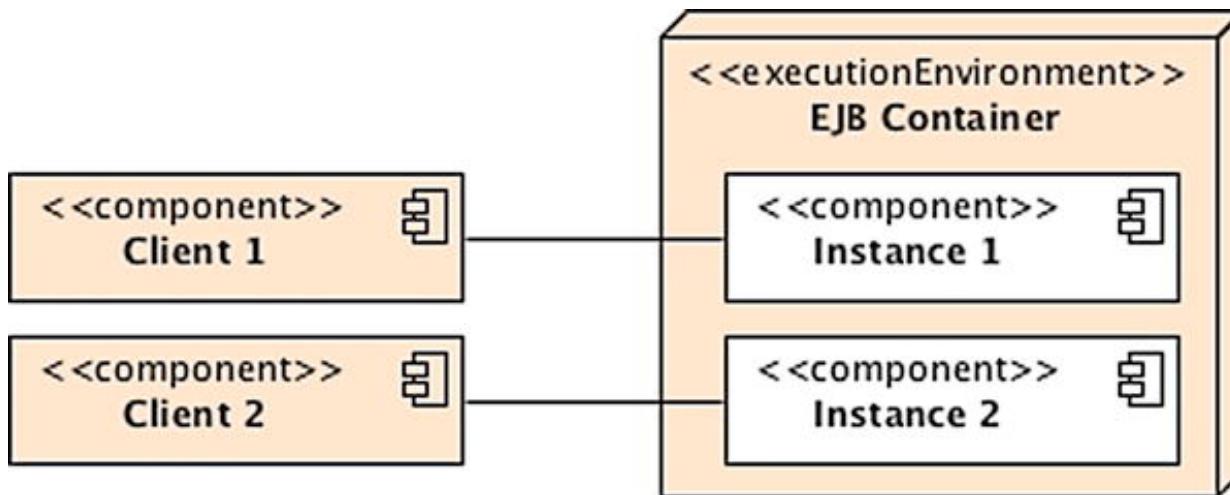
```
@Stateless  
public class ItemEJB {  
  
    @PersistenceContext(unitName = "chapter07PU")  
    private EntityManager em ;  
  
    public List<Book> findBooks() {  
        TypedQuery<Book> query = em.createNamedQuery(Book.FIND_ALL, Book.class);  
        return query.getResultList();  
    }  
  
    public List<CD> findCDs() {  
        TypedQuery<CD> query = em.createNamedQuery(CD.FIND_ALL, CD.class);  
        return query.getResultList();  
    }  
  
    public Book createBook(Book book) {  
        em.persist(book);  
        return book;  
    }  
  
    public CD createCD(CD cd) {  
        em.persist(cd);  
        return cd;  
    }  
}
```

Las llamadas a los métodos de negocio se encierran en transacciones

Enterprise Java Beans

Session Beans: Stateful

- Los EJB con estado guardan información entre las llamadas a sus métodos de negocio
- Como consecuencia, el contenedor crea una instancia nueva por cada cliente del EJB
- No es necesario hacerlo seguro para hebras porque cada cliente tiene su propia instancia
- El contenedor puede “pasivarlo” (guardarlo en disco)



Enterprise Java Beans

Session Beans: Stateful

■ Ejemplo

```
@Stateful
@StatefulTimeout(value = 20, unit = TimeUnit.SECONDS)
public class ShoppingCartEJB {

    private List<Item> cartItems = new ArrayList<>();

    public void addItem(Item item) {
        if (!cartItems.contains(item))
            cartItems.add(item);
    }

    public void removeItem(Item item) {
        if (cartItems.contains(item))
            cartItems.remove(item);
    }

    public Integer getNumberOfItems() {
        if (cartItems == null || cartItems.isEmpty())
            return 0;
        return cartItems.size();
    }

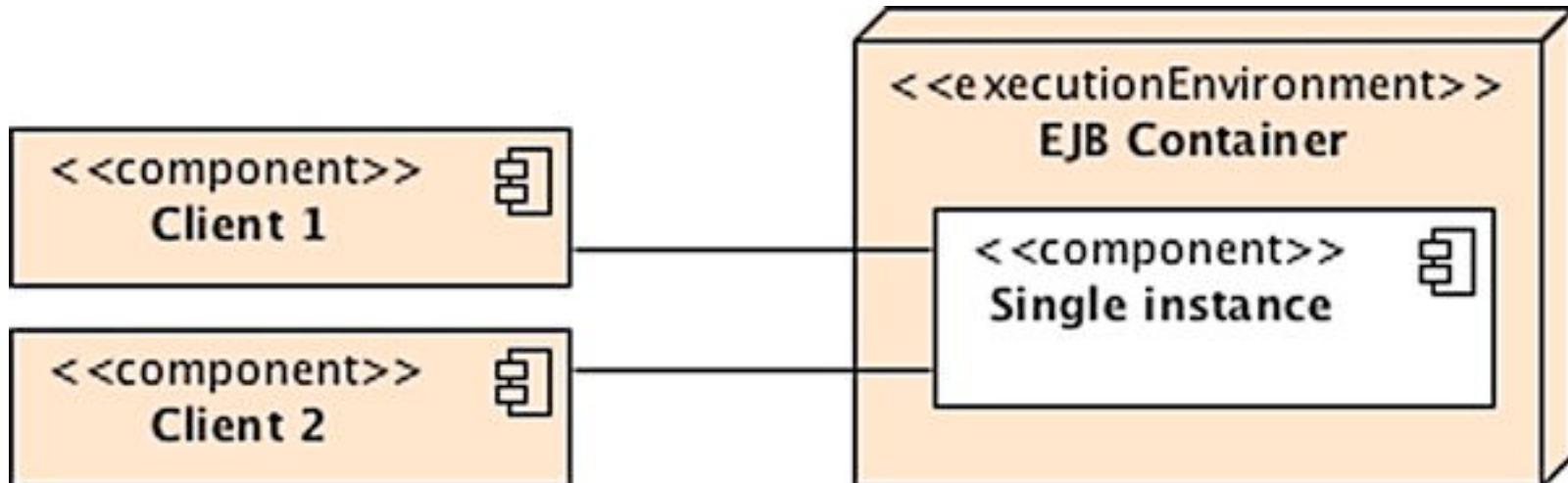
    public Float getTotal() {
        if (cartItems == null || cartItems.isEmpty())
            return 0f;

        Float total = 0f;
        for (Item cartItem : cartItems) {
            total += (cartItem.getPrice());
        }
        return total;
    }
}
```

Enterprise Java Beans

Session Beans: Singleton

- Solo existe un EJB singular de cada clase en una aplicación
- El contenedor se asegura de crear uno solo y compartirlo con todos los clientes que lo necesiten
- Puede haber accesos concurrentes a sus métodos y el desarrollador debe implementarlo *thread-safe*



Enterprise Java Beans

Session Beans: Singleton

Ejemplo

```
@Singleton
public class CacheEJB {

    private Map<Long, Object> cache = new HashMap<>();

    public void addToCache(Long id, Object object) {
        if (!cache.containsKey(id))
            cache .put(id, object);
    }

    public void removeFromCache(Long id) {
        if (cache.containsKey(id))
            cache .remove(id);
    }

    public Object getFromCache(Long id) {
        if (cache.containsKey(id))
            return cache .get(id);
        else
            return null;
    }
}
```

Enterprise Java Beans

Session Beans: acceso a un EJB

- Existen dos mecanismos para obtener una referencia a un EJB
 - Inyección de dependencias

Anotación clásica

```
// Client code injecting a reference to the EJB  
@EJB ItemEJB itemEJB;
```

Anotación CDI

```
// Client code injecting the produced remote EJB  
@ Inject ItemRemote itemEJBRemote;
```

- Búsqueda por JNDI (necesario si estamos fuera de un contenedor)

```
Context ctx = new InitialContext();  
ItemRemote itemEJB = (ItemRemote) →  
    ctx.lookup("java:global/cdbookstore  
/ItemEJB!org.agoncal.book.javaee7.ItemRemote");
```

Enterprise Java Beans

Session Beans: acceso a un EJB

- Podemos injectar cualquiera de las interfaces del EJB

```
@Stateless  
@Remote( ItemRemote.class )  
@Local( ItemLocal.class )  
@LocalBean  
public class ItemEJB implements ItemLocal , ItemRemote {...}  
  
// Client code injecting several references to the EJB or interfaces  
@EJB ItemEJB itemEJB;  
@EJB ItemLocal itemEJBLocal; ←  
@EJB ItemRemote itemEJBRemote;
```

No-interfaz

- Así es como inyectaremos una referencia en los backing beans

Enterprise Java Beans

Inyección de dependencias

- La inyección de dependencias es un servicio ofrecido por los contenedores que evitan tener que buscar objetos mediante JNDI
- Es posible injectar muchos tipos de recursos y con CDI es posible injectar POJOs también
- Anotaciones:
 - **@EJB**: para injectar EJBs
 - **@PersistenceContext**: para injectar contexto de persistencia
 - **@Inject**: para injectar casi todo

Enterprise Java Beans

Inyección de dependencias

```
@Stateless  
public class ItemEJB {  
  
    @PersistenceContext (unitName = "chapter07PU")  
    private EntityManager em;  
  
    @EJB  
    private CustomerEJB customerEJB;  
  
    @Inject  
    private NumberGenerator generator;  
  
    @WebServiceRef  
    private ArtistWebService artistWebService;  
  
    private SessionContext ctx;  
  
    @Resource  
    public void setCtx(SessionContext ctx) {  
        this.ctx = ctx;  
    }  
  
    //...  
}
```

También se puede aplicar a los setters

Enterprise Java Beans

Contexto del EJB

- Si queremos acceder al contexto del EJB para tener un control a más bajo nivel de los servicios ofrecidos por el contenedor inyectamos un recurso SessionContext

```
@Stateless  
public class ItemEJB {  
  
    @PersistenceContext(unitName = "chapter07PU")  
    private EntityManager em;  
  
    @Resource  
    private SessionContext context ;
```

Enterprise Java Beans

Contexto del EJB

- Algunos métodos de SessionContext son

Method	Description
getCallerPrincipal	Returns the <code>java.security.Principal</code> associated with the invocation.
getRollbackOnly	Tests whether the current transaction has been marked for rollback.
getTimerService	Returns the <code>javax.ejb.TimerService</code> interface. Only stateless beans and singletons can use this method. Stateful session beans cannot be timed objects.
getUserTransaction	Returns the <code>javax.transaction.UserTransaction</code> interface to demarcate transactions. Only session beans with bean-managed transaction (BMT) can use this method.
isCallerInRole	Tests whether the caller has a given security role.
lookup	Enables the session bean to look up its environment entries in the JNDI naming context.
setRollbackOnly	Allows the bean to mark the current transaction for rollback.
wasCancelCalled	Checks whether a client invoked the <code>cancel()</code> method on the client <code>Future</code> object corresponding to the currently executing asynchronous business method.

Enterprise Java Beans

Descriptor XML

- Como todos los componentes de Java EE que hemos visto hasta ahora, EJB tienen un descriptor XML que reemplaza a las anotaciones
- Su nombre es **ejb-jar.xml**

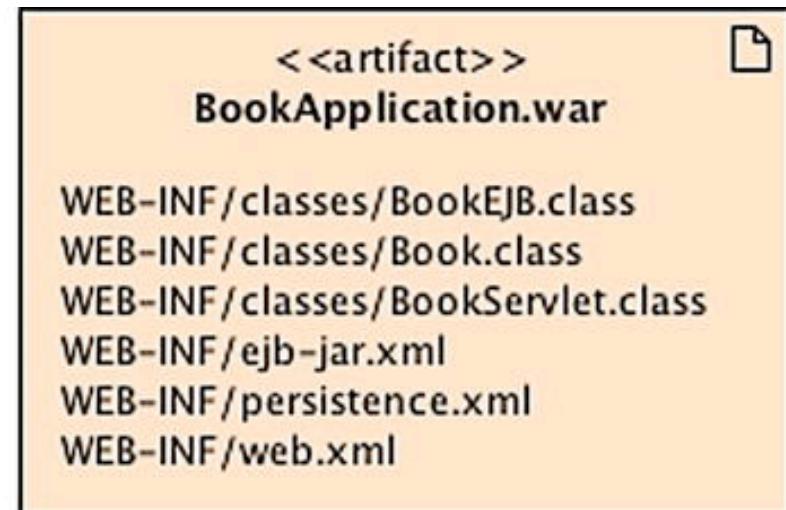
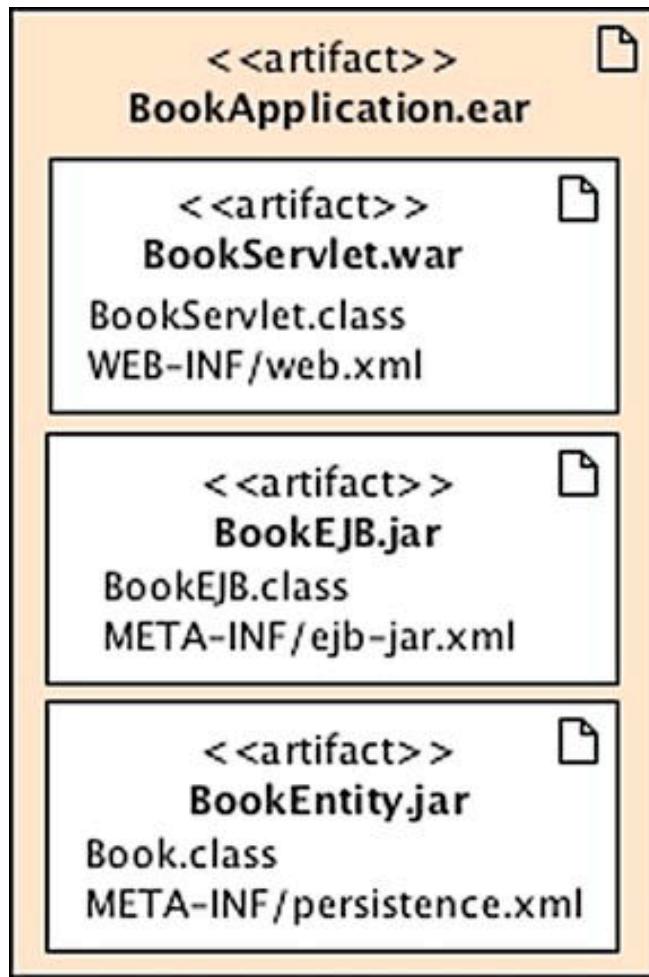
Enterprise Java Beans

Empaquetado

- En Java SE las clases se pueden empaquetar en archivos de extensión **jar** (zip)
- Las aplicaciones Web (servlets, JSP, JSF) se empaquetan en ficheros de extensión **war**
- Los EJBs se empaquetan en ficheros de extensión **jar**
- Las aplicaciones empresariales que tienen más de un módulo se empaquetan en ficheros de extensión **ear**

Enterprise Java Beans

Empaquetado



En Netbeans cada módulo será
un subproyecto diferente

Enterprise Java Beans

Temporizador

- El contenedor EJB posee un servicio de notificación basado en temporizador (*Timer Service*)
- Ejemplo:

El método se invoca cuando indica la anotación @Schedule

```
@Stateless
public class StatisticsEJB {

    @Schedule(dayOfMonth = "1", hour = "5", minute = "30")
    public void statisticsItemsSold() {
        // ...
    }

    @Schedules({
        @Schedule(hour = "2"),
        @Schedule(hour = "14", dayOfWeek = "Wed")
    })
    public void generateReport() {
        // ...
    }

    @Schedule(minute = "*/10", hour = "*", persistent = false )
    public void refreshCache() {
        // ...
    }
}
```

Stateless y
Singleton solo

Para ampliar conocimientos

- Antonio Goncalves, Beginning Java EE 7 (cap. 7, 8, 9 y 2)
- Especificación de EJB 3.2 (JSR 345):
<https://jcp.org/en/jsr/detail?id=345>
- Oracle Java EE 7 Tutorial (parte VII, caps. 32-36; parte II, caps.3-5)