Universidad de los Andes

Lógica de Negocio
Enterprise Java Beans – EJB 3.1

Por: Rafael Gustavo Meneses M.Sc.

Departamento de Ingeniería de Sistemas y Computación
Especialización en Construcción de Software
Bogotá, COLOMBIA

Universidad de los Andes

Agenda

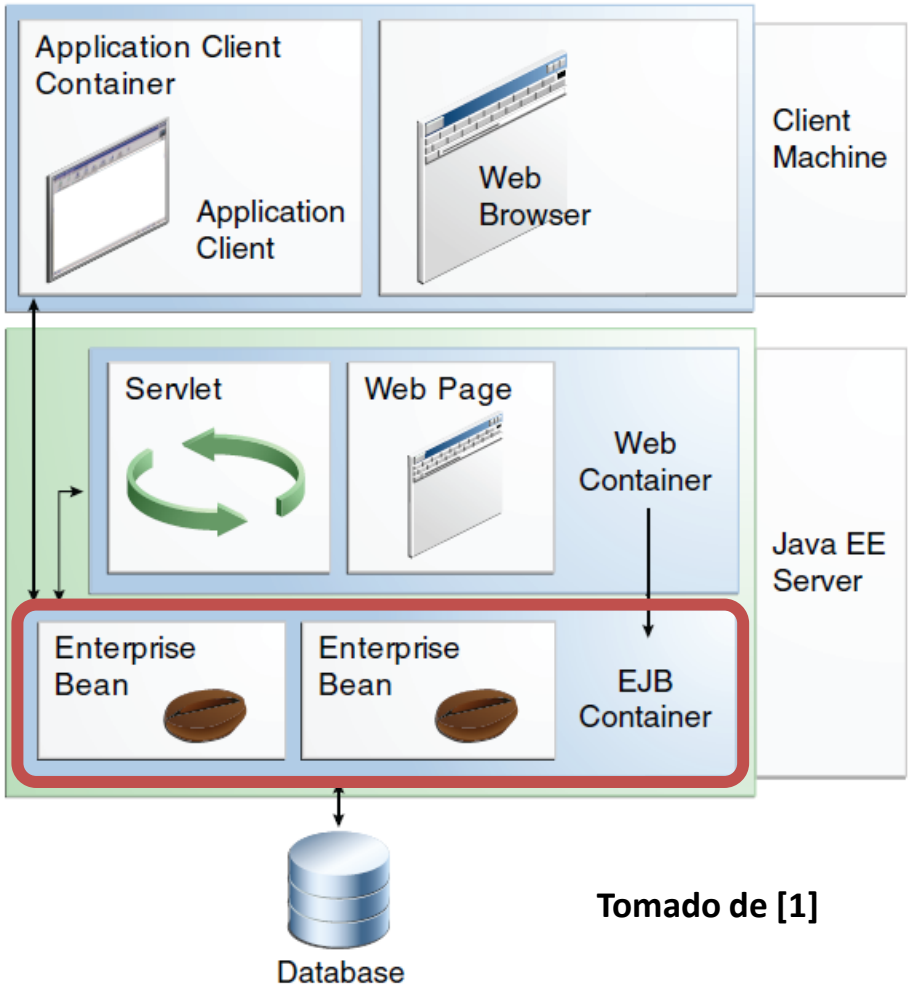
- Introducción
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- Session Beans
  - Acceso
  - Estado conversacional
  - Métodos de *Callback*
  - Localización
- Referencias

Universidad de los Andes

Agenda

- **Introducción**
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- Session Beans
  - Acceso
  - Estado conversacional
  - Métodos de *Callback*
  - Localización
- Referencias

Servidor Java EE y Contenedores

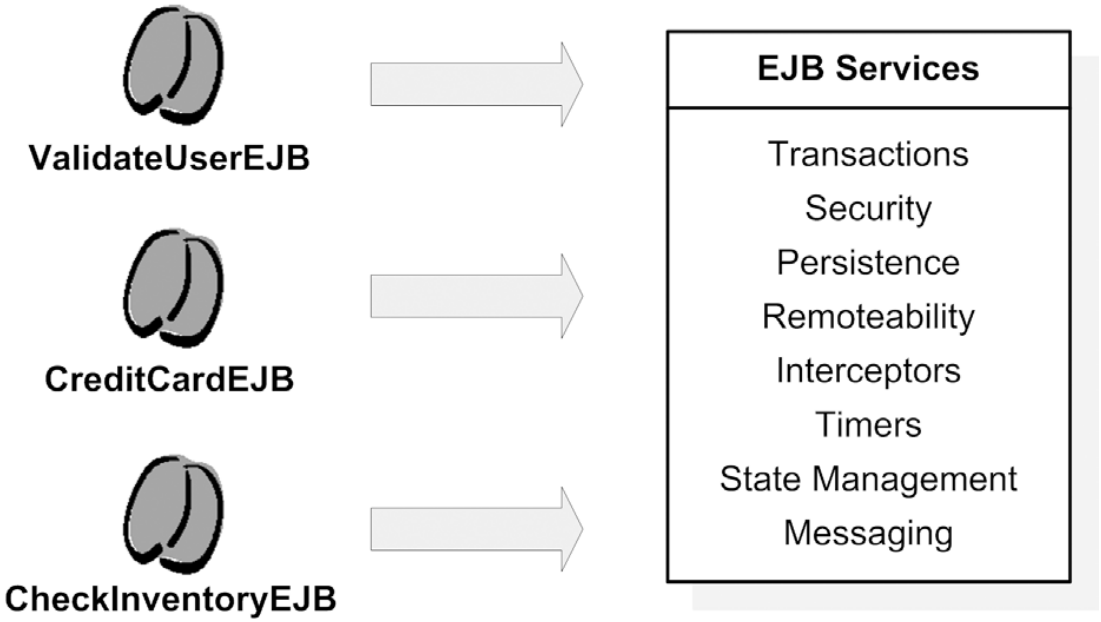


Tomado de [1]

Contenedor EJB

- Soporta concurrencia
- Provee pools para administrar varias instancias de componentes
- Balanceo de carga y clustering
- Provee servicio de nombres (JNDI) para acceder a los EJB o a otros recursos
- Soporta Java RMI-IIOP (Remote Method Invocation run over Internet Inter-Orb Protocol), el cual permite el acceso remoto de un cliente a un componente
- Soporta mensajería que proveen los message-driven beans

Contenedor EJB



Tomado de [2]

Universidad de los Andes

Agenda

- Introducción
- **Enterprise Java Beans (EJBs)**
  - Características
  - Beneficios
  - Tipos de componentes
- Session Beans
  - Acceso
  - Estado conversacional
  - Métodos de *Callback*
  - Localización
- Referencias

Definición

- *"Enterprise JavaBeans is an architecture for component-based transaction-oriented enterprise applications."* **[3]**
- *"An enterprise bean is a server-side component that encapsulates the business logic of an application."* **[1]**
- *"Es una plataforma para construir aplicaciones de negocios portables, reutilizables y escalables usando lenguaje de programación JAVA."* **[2]**

Características

- Contienen lógica de negocio, que opera sobre los datos de la empresa
- Las instancias de un EJB son administradas en tiempo de ejecución por un contenedor
- Pueden ser personalizados en tiempo de ejecución, editando sus variables de ambiente
- Servicios, tales como transaccionalidad y seguridad, pueden ser especificados:
  - En la lógica del negocio de la clase *enterprise bean* en forma de anotaciones
  - En un descriptor de despliegue XML

Características (II)

- El acceso del cliente es mediado por el contenedor en el cual el *enterprise bean* es desplegado. Este acceso es transparente para el cliente
- El contenedor asegura que los beans y sus clientes puedan ser desplegados en múltiples ambientes de ejecución sin re-compilación
- El estándar EJB 3 es desarrollado por ***Java Community Process (JCP)***

Beneficios

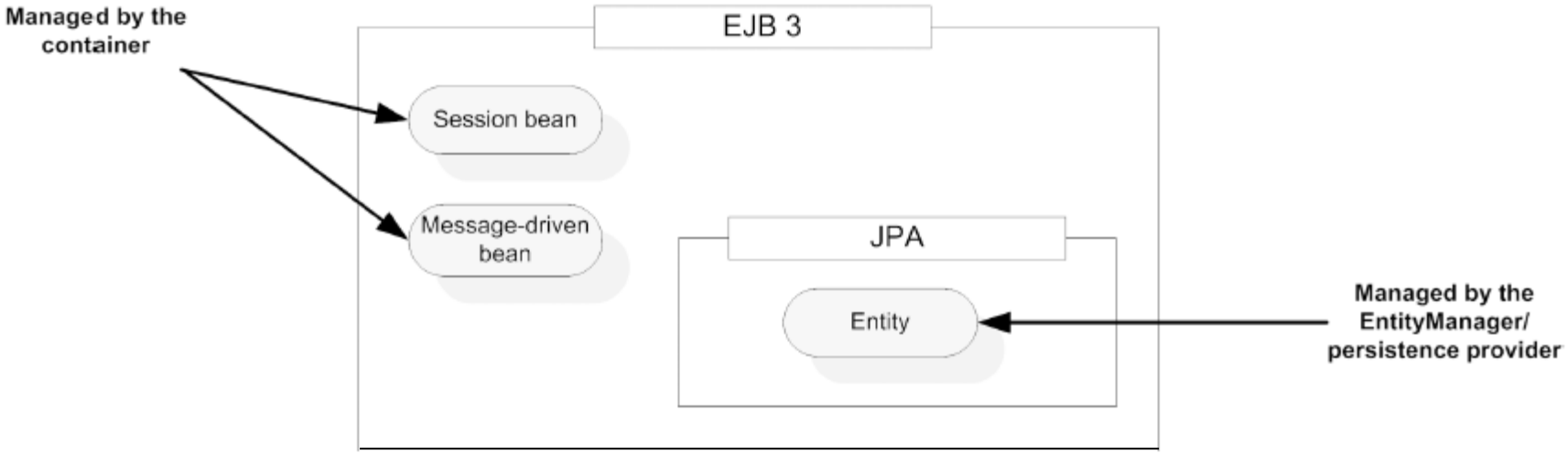
- *Simplifican el desarrollo* → El contenedor EJB es responsable de la administración de servicios a nivel del sistema (i.e. transacciones o autorizaciones de seguridad)
- *La lógica del negocio reside en los enterprise beans y no en el lado del cliente* → Permite que el desarrollo del lado del cliente esté desacoplado de la lógica del negocio
- *Son componentes* → Portables, reutilizables y pueden ser desplegados en servidores que usen los estándares del API JEE
- *Acceso Remoto* → Pueden residir en diferentes servidores y pueden ser invocados por un cliente remoto

¿Cuándo utilizarlos?

- *Aplicaciones que deben ser escalables* → Esto implica distribución de componentes a través de múltiples máquinas
- *Las transacciones deben asegurar integridad de los datos* → Los EJBs soportan transaccionalidad (mecanismo que administra el acceso concurrente de objetos compartidos)
- *Muti-usuarios locales y remotos*

Tipos de Componentes EJB

Session Beans	Stateful Session Beans
	Stateless Session Beans
	Singleton Session Beans
Message-Driven Beans	

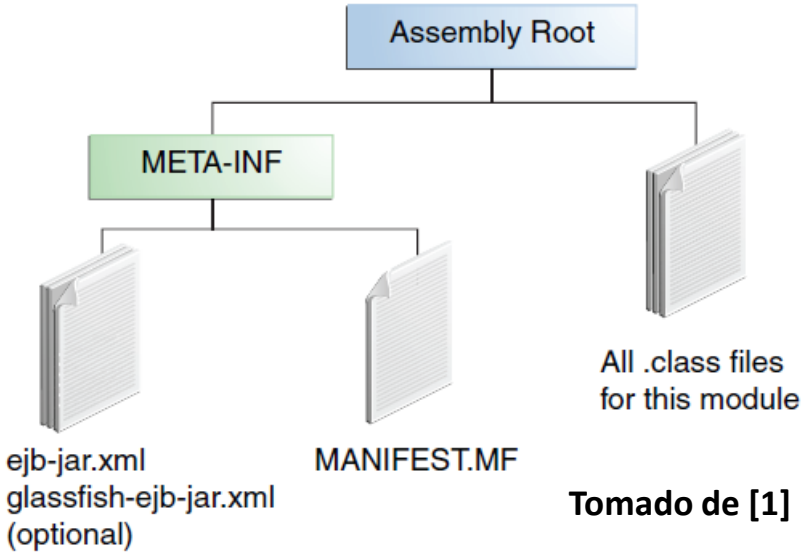


Tomado de [2]

Contenido de un EJB – Estructura de Módulos

- Un EJB debe contener:
  - *Clase enterprise bean* → Implementa los métodos de negocio y los métodos del ciclo de vida del EJB
  - *Interfaces de negocio* → Definen los métodos de negocio implementados por la clase enterprise bean
  - *Helpers* → Clases utilitarias requeridas por los enterprise bean

Se empaquetan en un archivo JAR EJB, son portables y pueden ser empaquetados en un archivo EAR



Tomado de [1]

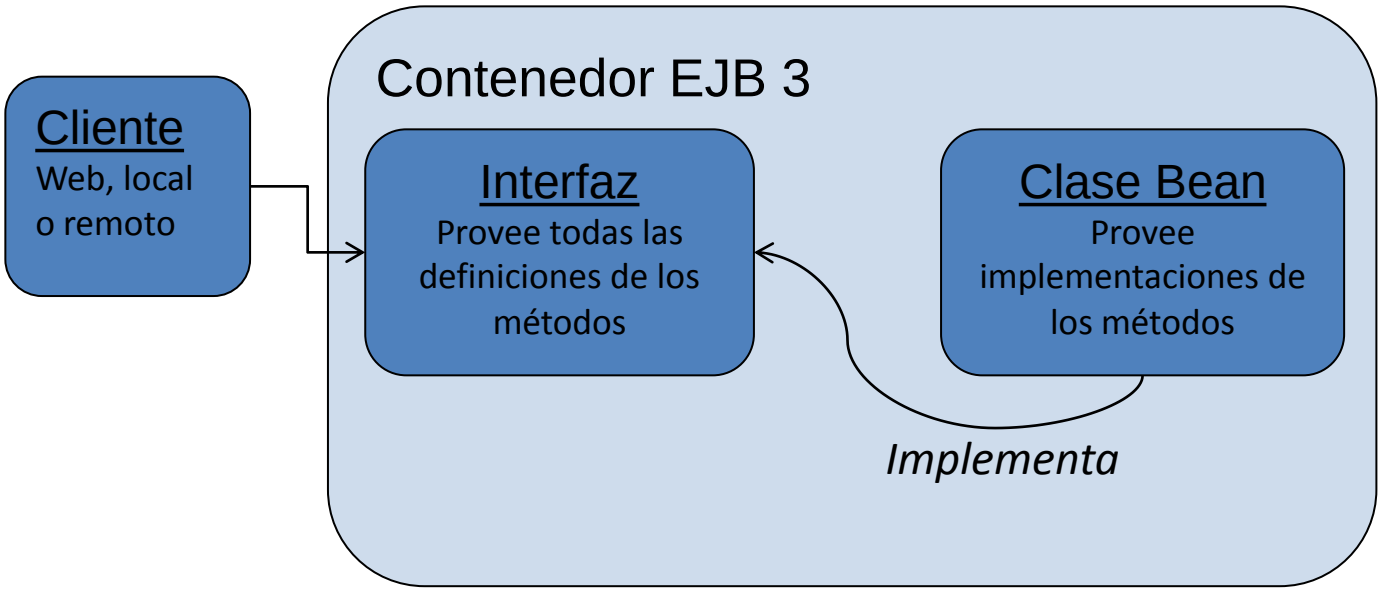
Universidad de los Andes

Agenda

- Introducción
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- **Session Beans**
  - Acceso
  - Estado conversacional
  - Métodos de *Callback*
  - Localización
- Referencias

Definición

- *"A session bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views."* **[1]**
- "Son una tecnología EJB que permiten encapsular procesos de negocio" **[3]**



Características

- Vida corta, si el servidor falla la sesión se pierde
- No manejan persistencia
- No es compartido entre clientes
- Pueden actualizar y crear entidades, estas últimas son persistentes
- Un cliente (local o remoto) interactúa con un *session bean* a través de la invocación de sus métodos (esta invocación se llama sesión)
- Un componente *session bean* es un POJO anotado

Tomado de [2]

Universidad de los Andes

Agenda

- Introducción
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- **Session Beans**
  - **Acceso**
  - Estado conversacional
  - Métodos de *Callback*
  - Localización
- Referencias

Acceso a un Session Bean

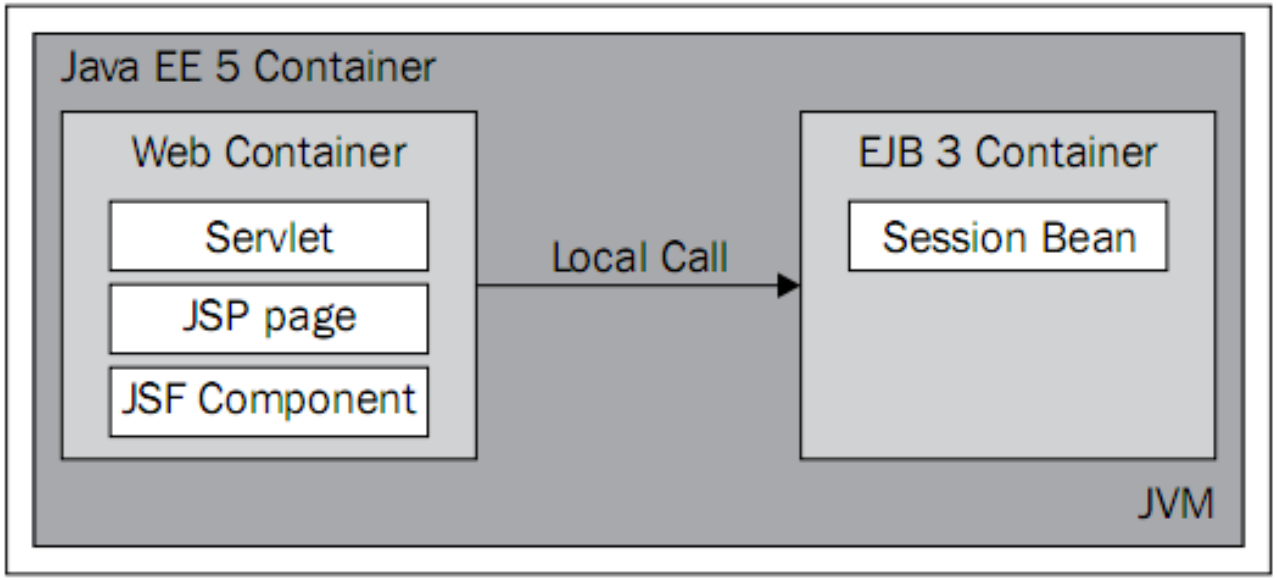
- Un *session bean* está compuesto por una o más interfaces y una clase de implementación
- Un cliente puede acceder a un *session bean* solamente a través de métodos definidos en la *interfaz* del bean
- La interfaz define la vista al cliente de un bean
- Un *session bean* puede ser invocado a través de RMI por medio de una interfaz:
  - Remota
  - Local

¿Acceso Local o Remoto?

- *Alto o bajo acoplamiento de beans relacionados* → Beans altamente acoplados son candidatos a ser accedidos localmente
- *Tipo de cliente* → Si el bean es accedido por *aplicaciones cliente*, éste debe permitir acceso remoto
- *Distribución de componentes* → En un escenario distribuido, los beans deben permitir acceso remoto
- *Desempeño*
  - Invocaciones remotas normalmente son más lentas que las invocaciones locales
  - Distribuir componentes en diferentes máquinas mejora el desempeño de toda la aplicación

Acceso Local

- Un cliente local puede invocar un *session bean* a través de una interfaz local. En este caso el cliente reside en el mismo lugar que la instancia del *session bean*



Tomado de [4]

Cliente Local

- Debe ejecutarse en la misma JVM en la cual se ejecutan los EJB a los que accede
- Puede ser un componente web u otro EJB
- La interfaz local define:
  - Los métodos de negocio del bean
  - Los métodos del ciclo de vida el ciclo de vida del bean
- La interfaz de un *session bean* por defecto es local

Implementación

- Crear una **clase** enterprise bean que **no** implementa una interfaz de negocio, indicando que el bean expone una vista *no-interface* al cliente

```
@Session
public class MyBean { ... }
```

- Anotar la interfaz de negocio del EJB como una interfaz **@Local**

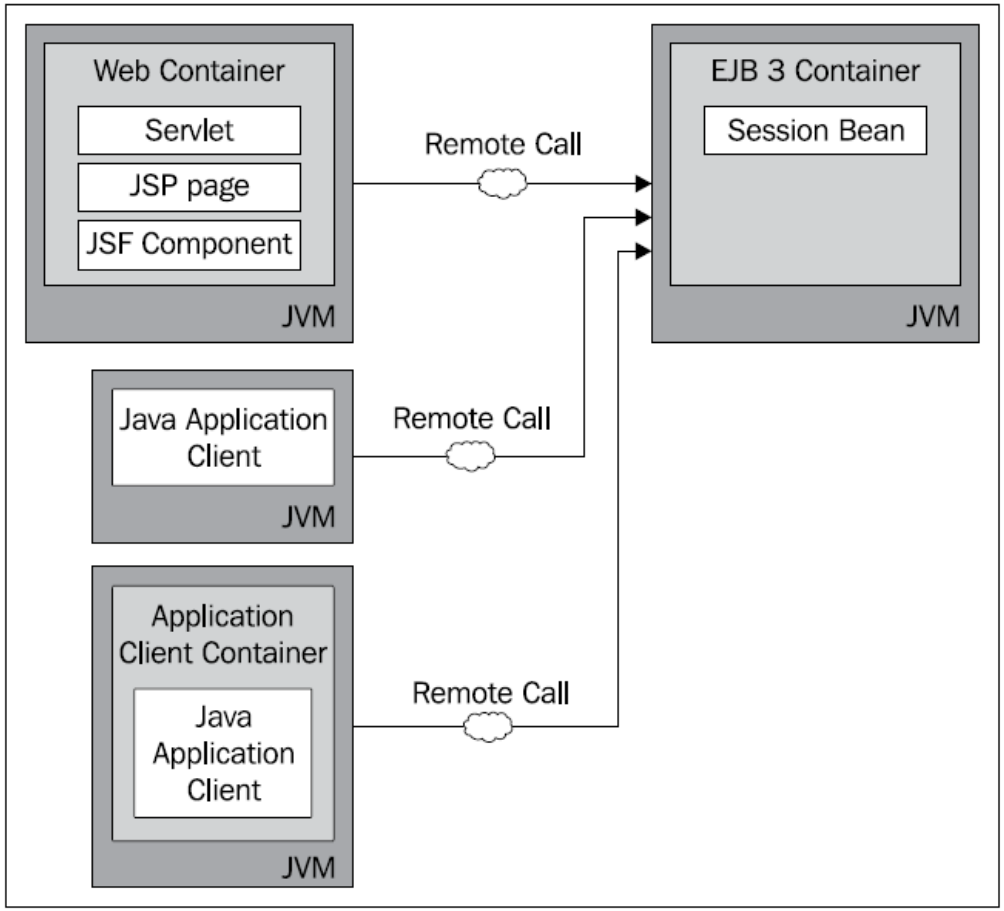
```
@Local
public interface InterfaceName { ... }
```

- Especificar la interfaz decorando la clase del bean con y especificando el nombre de la interfaz

```
@Local(InterfaceName.class)
public class BeanName implements InterfaceName { ... }
```

Acceso Remoto

- Un cliente remoto invoca una interfaz remota de un *session bean*



Tomado de [4]

Cliente Remoto

- Puede ejecutarse en una máquina diferente y una JVM diferente al *enterprise bean* que accede
- Puede ser un componente web, una aplicación cliente u otro *enterprise bean*
- Para el cliente remoto la ubicación del *enterprise bean* es transparente
- La interface remota define los métodos de negocio y del ciclo de vida del *session bean*
- El *enterprise bean* **debe** implementar una interfaz de negocio
  - Clientes remotos **no** pueden acceder un bean a través de una vista no-interfaz

Implementación

- Decorar la interfaz de negocio del bean con **@Remote**

```
@Remote
public interface InterfaceName { ... }
```

- Decorar la clase del bean con **@Remote**, especificando la interfaz de negocio

```
@Remote(InterfaceName.class)
public class BeanName implements InterfaceName { ... }
```

Universidad de los Andes

Agenda

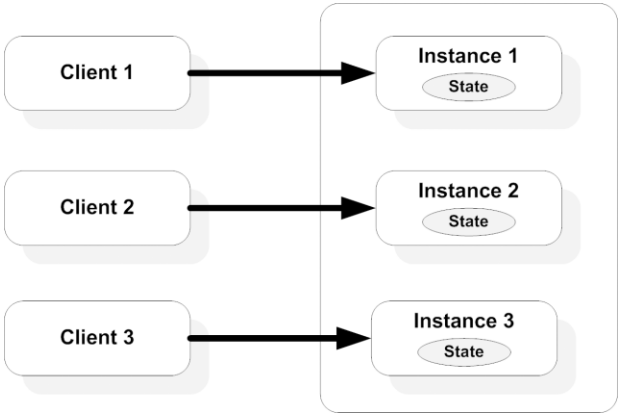
- Introducción
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- **Session Beans**
  - Acceso
  - **Estado conversacional**
  - Métodos de *Callback*
  - Localización
- Referencias

Estado Conversacional

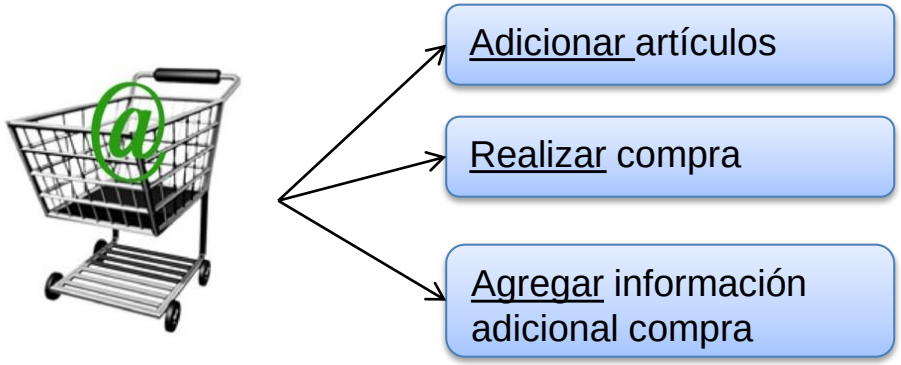
- El estado de un objeto se compone de los valores de sus variables de instancia
- Las instancias de las variables representan el estado de una única sesión *cliente-bean*
- El estado de la interacción del cliente con el bean es llamado *estado conversacional*
- Tipos de session beans:
  - Stateful
  - Stateless
  - Singleton

Stateful Session Beans

- El estado se **mantiene** durante la sesión del cliente con el bean
- Una instancia es reservada para un cliente y cada una almacena la información de dicho cliente



- La sesión finaliza si el cliente remueve el bean o finaliza su sesión



Stateful Session Beans

Implementación

```
@Stateful
public class ShoppingCartBean implements ShoppingCart {
    private ArrayList<String> items;
    @PostConstruct
    public void initialize() {
        items = new ArrayList<String>();
    }
    public void addItem(String item) {
        items.add(item);
    }
    public Collection<String> getItems() {
        return items;
    }
    @Remove
    public void finished() {
        System.out.println(
                "Remove method finished() Invoked");
    }
}
```

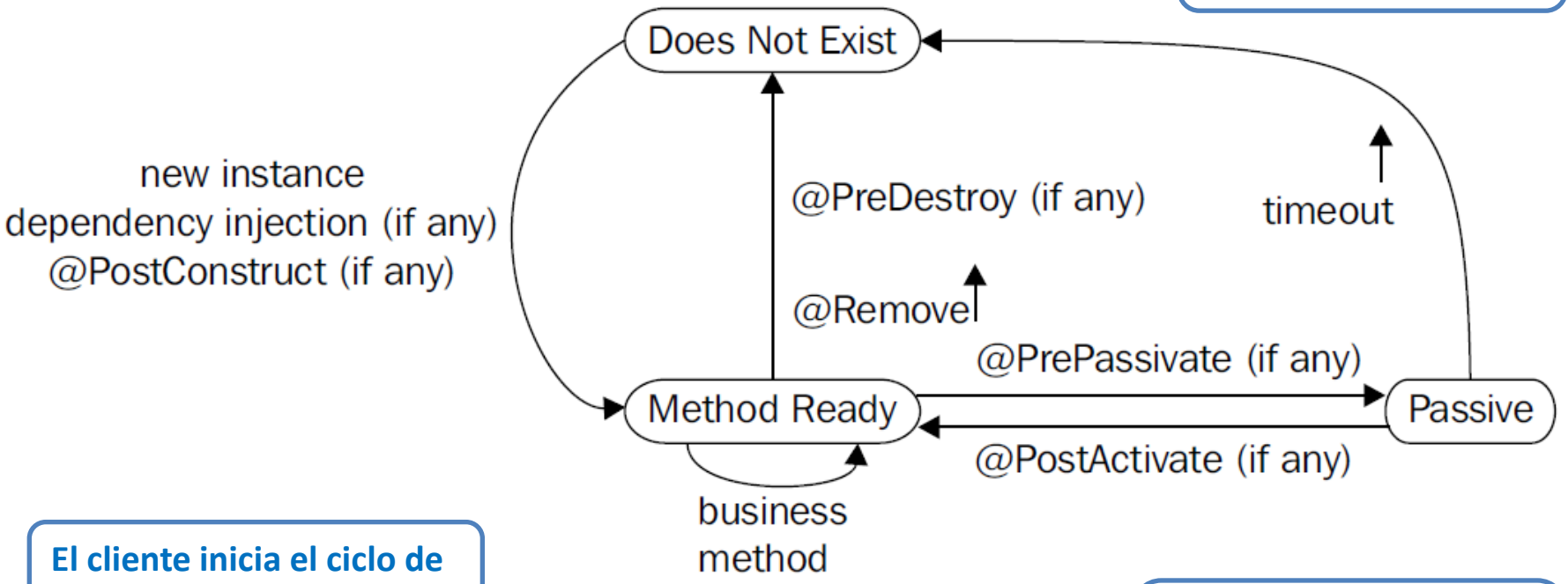
```
@Remote
public interface ShoppingCart {
    public void initialize();
    public void addItem(String item);
    public Collection<String> getItems();
    public void finished();
}
```

Stateful Session Beans

Ciclo de Vida

El método *remove* es el único que es invocado directamente por el cliente

El contenedor decide desactivar el bean

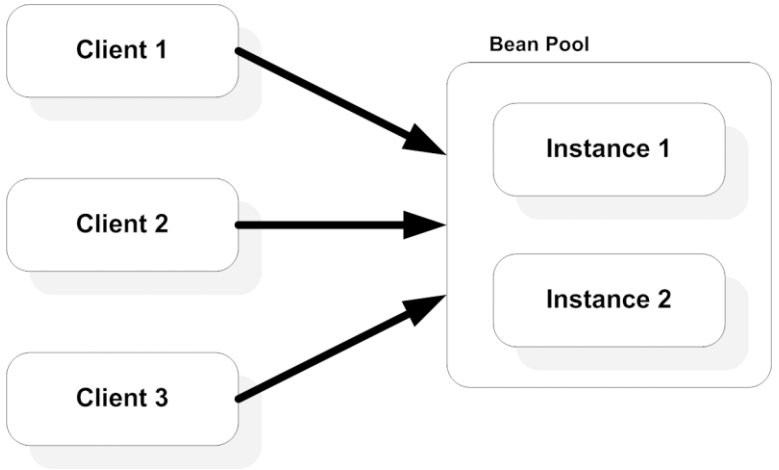


El cliente inicia el ciclo de vida obteniendo una referencia al session bean stateful

El cliente invoca algún método del negocio durante la desactivación

Stateless Session Beans

- No mantiene un *estado conversacional* con el cliente
  - Cuando un cliente invoca los métodos de un stateless bean, las variables de instancia del bean pueden contener un estado específico del cliente, pero sólo por la duración de la invocación.
  - Cuando el método finaliza, el estado del cliente específico no debería mantenerse
- Las instancias pueden estar compartidas por los clientes. El contenedor tiene un pool de instancias, cuando el cliente invoca un método se asigna una instancia, cuando la libere es retornada al pool
- Las instancias pueden estar compartidas por los clientes
- El contenedor tiene un pool de instancias, cuando el cliente invoca un método se asigna una instancia, cuando la libera es retornada al pool



Stateless Session Beans

Implementación

- Ofrecen mejor escalabilidad para aplicaciones con gran cantidad de clientes
- Puede implementar un *web service*

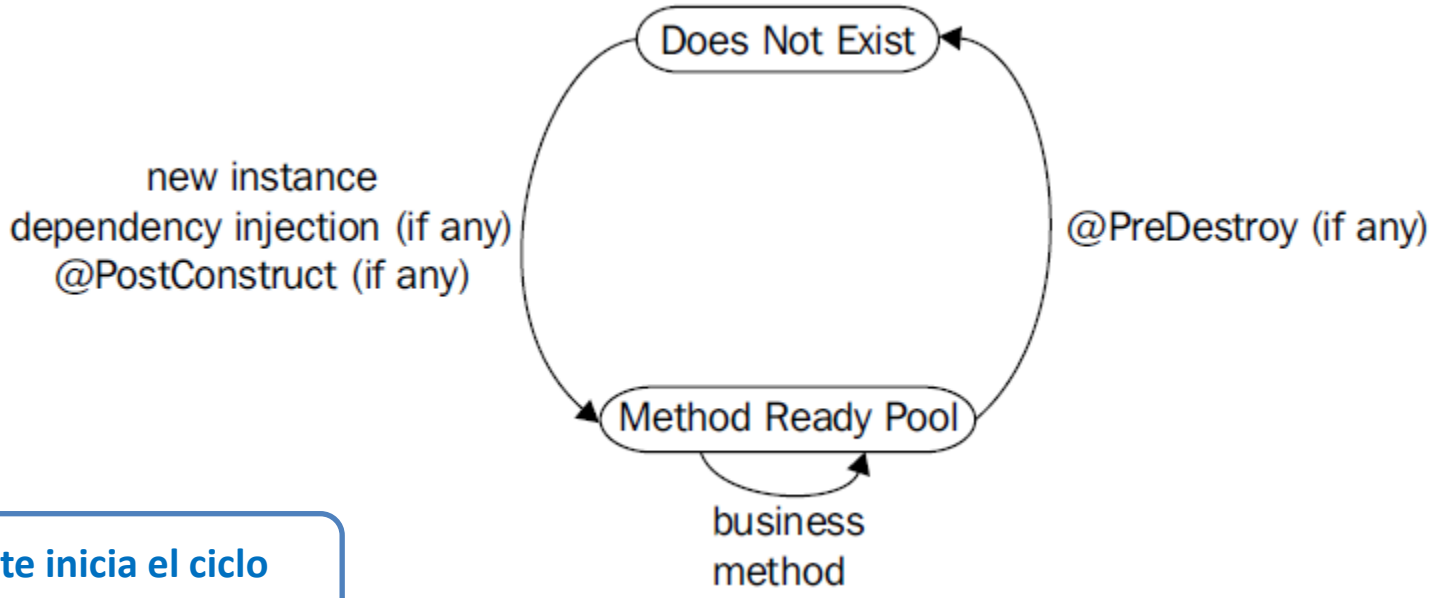
```
@Remote
public interface BidManager {
    void addBid(Bid bid);
    void cancelBid(Bid bid);
    List<Bid> getBids(Item item);
}
```

Implementación

```
@Stateless(name="BidManager")
public class BidManagerBean implements BidManager {
    @Resource(name="jdbc/ActionBazaarDS")
    private DataSource dataSource;
    private Connection connection;
    ...
    public BidManagerBean() {}
    public void addBid(Bid bid){
        try {
            Long bidId = getBidId();
            Statement statement = connection.createStatement();
            statement.execute(
                "INSERT INTO BIDS ("
                    + "BID_ID, "
                    + "BID_AMOUNT, "
                    + "BID_BIDDER_ID, "
                    + "BID_ITEM_ID) "
                    + "VALUES ("
                    + bidId + ", "
                    + bid.getAmount() + ", "
                    + bid.getBidder().getUserId() + ", "
                    + bid.getItem().getItemId()+ ")");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Stateless Session Beans

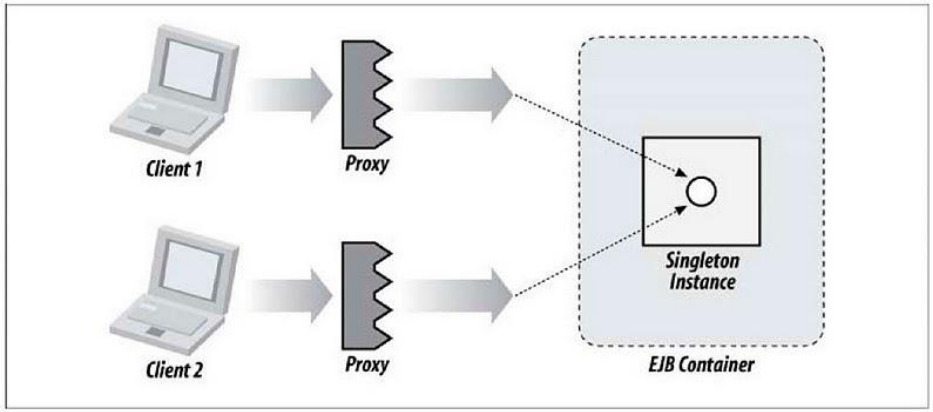
Ciclo de Vida



El cliente inicia el ciclo de vida obteniendo una referencia al bean de sesión stateless

Singleton Session Beans

- Enterprise bean instanciado una vez por aplicación
- Existe durante todo el ciclo de vida de la aplicación
- Son diseñados para casos en los que una misma instancia de un bean es *compartida* y *accedida concurrentemente* por los clientes de la aplicacion
- Similar a un *stateless sesion bean*
  - No mantiene un *estado conversacional* con el cliente
  - Puede implementar *end-points* de web services
- Puede ser utilizado para ejecutar tareas de inicialización de la aplicación o para tareas de limpieza cuando la aplicación termina su ejecución



Tomado de:
http://what-when-how.com/enterprise-javabeans-3-1/the-singleton-session-bean-enterprise-javabeans-3-1/

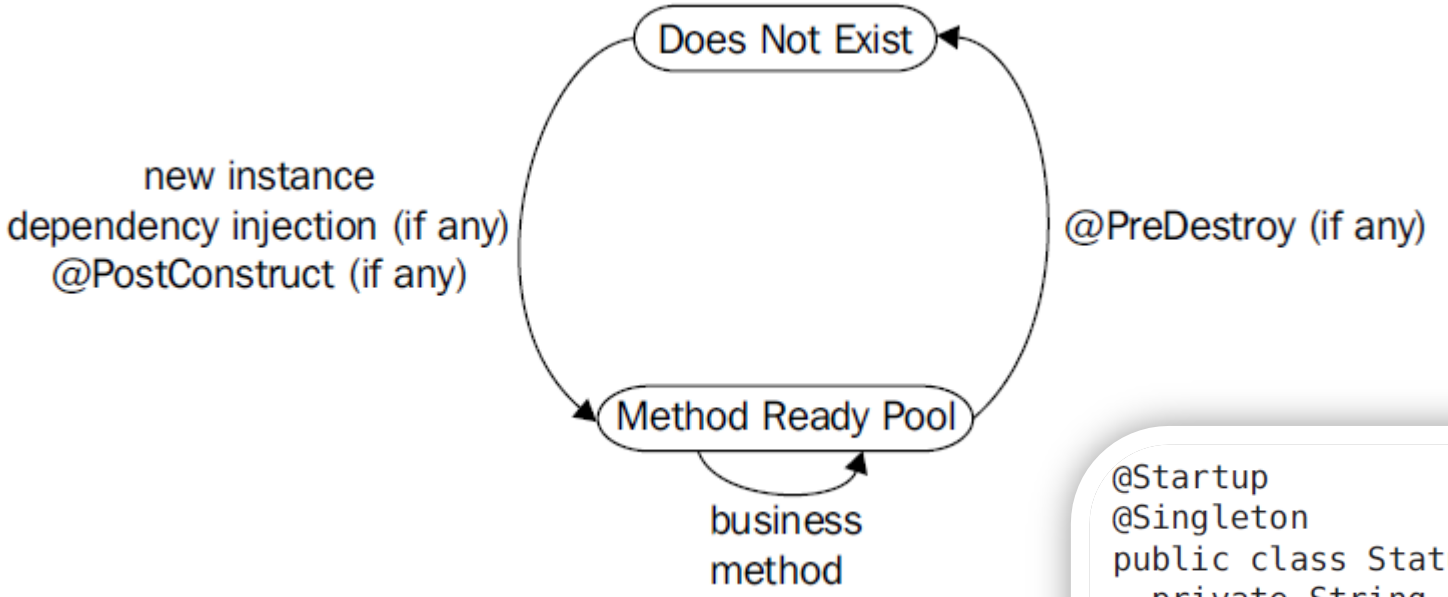
Singleton Session Beans

Implementación

```
package counter.ejb;

import javax.ejb.Singleton;
import javax.ejb.ConcurrencyManagement;
import static javax.ejb.ConcurrencyManagementType.CONTAINER;
import javax.ejb.Lock;
import javax.ejb.LockType.WRITE;

/**
 * CounterBean is a simple singleton session bean that records the number
 * of hits to a web page.
 */
@Singleton
@ConcurrencyManagement(CONTAINER)
public class CounterBean {
    private int hits = 1;

    // Increment and return the number of hits
    @Lock(WRITE)
    public int getHits() {
        return hits++;
    }
}
```

Singleton Session Beans

Ciclo de Vida



El contenedor es responsable por inicializar un bean singleton, sin embargo…

```
@Startup
@Singleton
public class StatusBean {
  private String status;

  @PostConstruct
  void init {
    status = "Ready";
  }
  ...
}
```

Agenda

- Introducción
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- **Session Beans**
  - Acceso
  - Estado conversacional
  - **Métodos de *Callback***
  - Localización
- Referencias

Métodos de Callback

- Los métodos *callback* son métodos del bean (no expuestos en la interfaz) que el contenedor *llama* para notificar la transición del ciclo de vida de un bean
- Cuando el evento del *ciclo de vida* del bean ocurre, el contenedor invoca al método callback correspondiente
- Estos métodos son marcados con anotaciones como **@PostContruct** y **@PreDestroy** y para los stateful session bean se agregan **@PrePassivate** y **@PostActivate**

Métodos de Callback (II)

- Deben ser públicos, sin retorno (void), y sin parámetros
- Utilizan las siguientes anotaciones:
  - **@PostConstruct** → Invocado sobre una instancia recientemente creada después de la inyección (o JNDI lookup) de todas las dependencias y antes de la invocación del primer método
  - **@PreDestroy** → Invocado luego de que un método anotado con @Remove ha terminado y antes de que el contenedor elimine la instancia del bean
  - **@PrePassivate** → Invocado antes de que el contenedor desactive (passivate) el bean, el contenedor deshabilita temporalmente el bean y lo guarda en memoria secundaria
  - **@PostActivate** → Invocado después de que el contenedor mueve el bean de memoria secundaria a estado activo (active)

Implementación

```
@Stateless(name="BidManager")
public class BidManagerBean implements BidManager {
    @Resource(name="jdbc/ActionBazaarDS")
    private DataSource dataSource;
    private Connection connection;
    ...
    public BidManagerBean() {}

    @PostConstruct
    public void initialize() {
        try {
          connection = dataSource.getConnection();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
    }
    ...
```

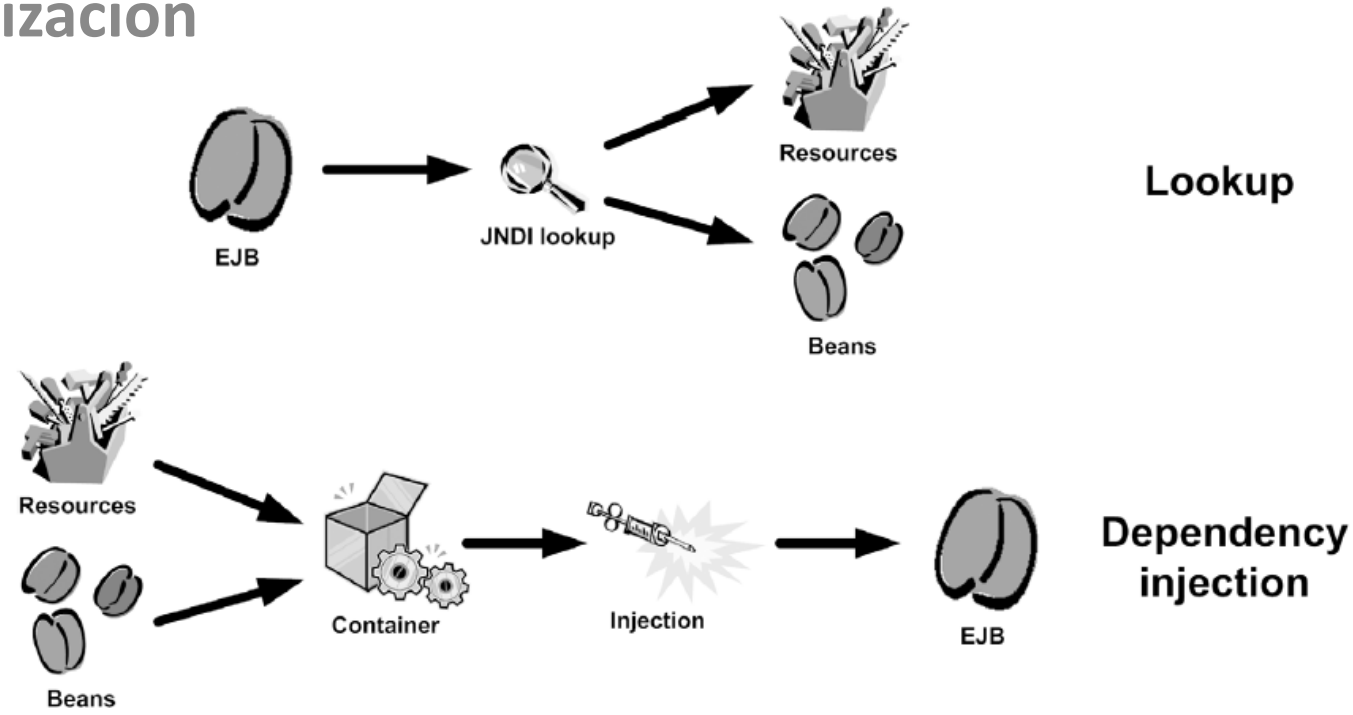
```
@PreDestroy
public void cleanup() {
        try {
            connection.close();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
}
```

Universidad de los Andes

Agenda

- Introducción
- Enterprise Java Beans (EJBs)
  - Características
  - Beneficios
  - Tipos de componentes
- **Session Beans**
  - Acceso
  - Estado conversacional
  - Métodos de *Callback*
  - **Localización**
- Referencias

Localización



- Con JNDI es responsabilidad del cliente hacer la localización y obtener la referencia al objeto
- Con EJB 3, se puede utilizar la *inyección de dependencias*, dejando que el contenedor se responsabilice de inyectar un objeto basado en la declaración de la inyección

Localización local – Por medio de la vista no-interface

- Utilizando la anotación **`javax.ejb.EJB`** en el cliente, especificando el nombre de la clase del enterprise bean:

```
@EJB
ExampleBean exampleBean;
```

- A través un de un *lookup JNDI*

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

- Los clientes nunca utilizan el operador ***new***

Localización local – Por medio de la interface de negocio

- Utilizando la anotación **`javax.ejb.EJB`** en el cliente, especificando el nombre de la interfaz de negocio *local*:

```
@EJB
Example example;
```

- A través un de un *lookup JNDI*

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

- Los clientes nunca utilizan el operador ***new***

Localización remota – Por medio de la interface de negocio

- Utilizando la anotación **`javax.ejb.EJB`** en el cliente, especificando el nombre de la interfaz de negocio *remota*:

```
@EJB
Example example;
```

- A través un de un *lookup JNDI*

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

- Los clientes nunca utilizan el operador ***new***

Implementación

```
public class Client {
    @EJB
    private static ShoppingCart shoppingCart;
    public static void main(String args[]) throws Exception {
        shoppingCart.addItem("Bread");
        shoppingCart.addItem("Milk");
        shoppingCart.addItem("Tea");
        System.out.println("Contents of your cart are:");
        Collection<String> items = shoppingCart.getItems();
        for (String item : items) {
            System.out.println(item);
        }
        shoppingCart.finished();
    }
}
```

Universidad de los Andes

JNDI (Java Naming and Directory Interface)

- Un componente Java EE puede localizar su contexto de nombres con JNDI
- Un componente puede crear un objeto *javax.naming.InitialContext* y buscar su contexto bajo el nombre de *java:comp/env*
- Un componente JEE puede acceder nombres provistos por el sistema (*named system-provided*) y objetos definidos por los usuarios (*user-defined objects*)
- Tres (3) namespaces JNDI namespaces pueden utilizarse para lookups
  - *java:global*
  - *java:module*
  - *java:app*

JNDI (Java Naming and Directory Interface)

- *java:global* se utiliza para encontrar enterprise beans remotos:

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

- *java:module* se utiliza para encontrar enterprise beans dentro del mismo módulo:

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

- *java:app* se utiliza para hacer lookup de enterprise beans empaquetados dentro de la misma aplicación:

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

Nuevas Características (con respecto a la versión 3.0)

- Singleton Session Beans
- The EJB Timer
- No-Interface Views
- Asynchronous Services
- Simplified Deployment
- CDI (Contexts and Dependency Injection)

Referencias

1. The Java™ EE 6 Tutorial. Eric Jendrock. Oracle Corporation. 2011.
2. EJB 3 in Action. Panda Debu, Rahman Reza, Lane Derek. Manning. 2007.
3. JSR-318 Enterprise JavaBeansTM 3.1 (Final Release). Sun Microsystems. 2009.
4. EJB 3 Developer Guide. Sikora, Michael. 2008.

Universidad de los Andes

Gracias !!!

Rafael Meneses

rg.meneses81@uniandes.edu.co