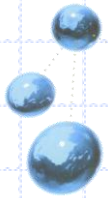


CC68J APLICACIONES EMPRESARIALES CON JEE

ENTERPRISE JAVA BEANS

2. Resumen de la arquitectura EJB

Profesores:

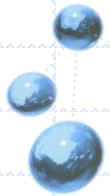
- Andrés Farías

fcfm

v1.0

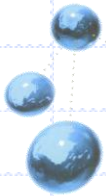
Objetivos de esta sección

1. Entity Beans (Beans de Entidad).
2. El Componente Enterprise Bean.
3. Session Beans.
4. El contrato del Contenedor EJB.



Beans de Entidad

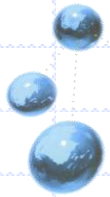
Entity Beans



Entity Beans

Conceptos Generales

- Los Bean de Entidad (Entity Beans) en Java Persistance 1.0 están disponibles sólo como POJOs, y son mapeados a tablas en una BDD relacional.
- A diferencia de otros tipos de EJB, los Entity pueden ser:
  - ✓ Asignados
  - ✓ Serializados
  - ✓ Enviados a través de la red.
- Un Entity Bean modela conceptos de negocio que pueden ser expresados como sustantivos. Por ejemplo:
  - ✓ Cliente
  - ✓ Pieza de equipo.
  - ✓ Item de un inventario.
  - ✓ Un lugar.



Diseño de Entity Beans

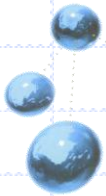
A través de la implementación

- Para implementar un Entity Bean hay que definir un bean y decidir qué campo será la llave primaria de ese bean:
- Llave primaria
  - ✓ La llave primaria provee un puntero a la BDD.
  - ✓ Le da identidad al Entity, tanto en memoria como objeto y como fila en una tabla.
  - ✓ La llave primaria puede ser un tipo complejo o primitivo.
- Clase Bean
  - ✓ El Entity Bean es la representación del objeto de la BDD.
  - ✓ La única lógica de negocio que debiera ir en este Bean es de validación respeto a sus estructura interna.
  - ✓ No debe implementar ninguna interfaz o ser serializable.
  - ✓ Debe ser *anotado* con `@javax.persistance.Entity`.
  - ✓ Debe identificar al menos un campo como llave primaria, anotándolo con `@javax.persistence.Id`.

Entity Manager

Administrador de los beans de entidad

- El **Entity Manager** es un servicio provisto por Java Persistance que permite interactuar con los Beans de Entidad:
  - ✓ Consultar Entities
  - ✓ Almacenar Entities.
  - ✓ Actualizar su información.
  - ✓ Etc.
- El Entity Manager provee una API de consulta, y métodos del ciclo de vida de un Entity Bean.
- En Java Persistance, Entity Beans y el Entity Manager no requieren de un servidor de aplicaciones para ser usados. Pueden ser usados como cualquier otro framework (Por ej: Hibernate).



La Clase Entity Bean

Definiendo una clase (1/2)

```
import javax.persistence.* ;

@Entity
@Table(name="CABIN")
public class Cabin {
    private int id;
    private String name;
    private int deckLevel;

    @Id
    @GeneratedValue
    @Column(name="CABIN_ID")
    public int getId( ) { return id; }
    public void setId(int pk) { this.id = pk; }
```

Anotación para hacerlo en un bean de Entidad.

Anotación para declarar la tabla a la que está mapeado el bean.

Estructura interna del bean

Anotación para indicar que el siguiente atributo es el identificador.

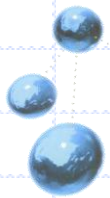
Indicar que el contenedor o la base de datos generará automáticamente el ID cuando nuevas instancias del bean sean creadas

La Clase Entity Bean

Definiendo una clase (2/2)

```
    ...

    @Column(name="CABIN_NAME")
    public String getName( ) { return name; }
    public void setName(String str) { this.name = str; }

    @Column(name="CABIN_DECK_LEVEL")
    public int getDeckLevel( ) { return deckLevel; }
    public void setDeckLevel(int level) { this.deckLevel = level; }
}
```

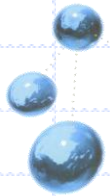
Anotación para indicar la columan de la tabla a la cual se mapea el atributo.



Descriptors y JAR's

Desplegando los Beans

- Descriptores de despliegue de persistencia.
  - ✓ Los Beans de Entidad son agrupados en un conjunto finito de clases llamados **Unidad de Persistencia** (**Persistance Unit**).
  - ✓ Cada Unidad de Persistencia debe estar asociada a una base de datos en particular.
  - ✓ El servicio EntityManager es responsable de administrar las unidades de persistencia, por lo que estas deben estar identificadas para poder ser referenciadas desde el código.
  - ✓ Toda esta información es almacenada en un descriptor de despliegue XML llamado **`persistance.xml`**, que es un artefacto requerido!
- Empaquetamiento:
  - ✓ Una vez que se han definido los descriptores de despliegue y las clases bean, todos deben ser empaquetados en un JAR.
  - ✓ El proveedor de persistencia examinará el JAR para determinar como desplegar una o más unidades de persistencia en el ambiente.

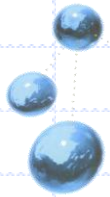


El componente Enterprise Bean

Componentes Enterprise Bean

Session Beans

- Características:
  - ✓ Son extensiones de la aplicación cliente que ejecutan procesos o tareas de negocio.
  - ✓ Pueden afectar directamente entidades de negocio, pero no necesitan conocer el contexto de dichas operaciones.
  - ✓ Relacionan diversas entidades de negocio.
- Ejemplo:
  - ✓ **Hacer una reservación**. Relaciona un Crucero, una Cabina y un Cliente.



Componentes Enterprise Bean

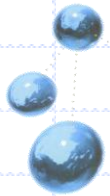
Message-driven Beans

- Características
  - ✓ Coordinan tareas que involucran Beans de Sesión y Beans de entidad.
  - ✓ Un message-driven bean utiliza el paradigma publicador-subscriptor para funcionar.
  - ✓ Responde procesando un mensaje y administrando las acciones que otros beans realizan.
- Ejemplo
  - ✓ Un proceso de reserva asíncrono, que recibe eventos provenientes desde un sistema legado, coordina las acciones correspondientes al proceso, involucrando los cruceros, cabinas, pasajeros, etc.

Diseño de Session y Message Beans

A través de la implementación

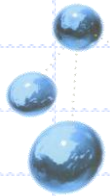
- Interfaz remota
  - ✓ Define los métodos de negocio de un bean de sesión, que pueden ser invocados desde fuera del contenedor.
  - ✓ Es tageada con la anotación `@javax.ejb.Remote`.
- Interfaz local
  - ✓ Define los métodos del Bean que pueden ser invocados por otros beans dentro del mismo contenedor.
  - ✓ Es tageada con la anotación `@javax.ejb.Local`.
- Interfaz de punto final (endpoint)
  - ✓ Define los métodos de negocio que pueden ser invocados desde el exterior del contendor EJB utilizando el protocolo SOAP.
  - ✓ Es tageada con la anotación `@javax.jws.WebService`.
- Interfaz de mensajes
  - ✓ Define los métodos de un Message-driven Bean que los sistemas de mensajería, tales como JMS, pueden entregarle al Bean.
- Clase Bean
  - ✓ Es la clase que contienen los métodos de negocio de un bean de sesión.
  - ✓ Debe tener al menos una de las interfaces: `Remote`, `Local` o `WebService`.
  - ✓ El bean debe tener el tag `@javax.ejb.Stateful` o `@javax.ejb.Stateless`.



Diseño de Session y Message Beans

Consideraciones

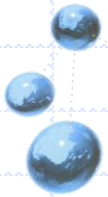
- Un message-driven bean implementa uno o más métodos de recepción de mensajes (`onMessage()` por ejemplo).
- El message-driven bean debe tener la anotación `@javax.ejb.MessageDriven`.
- Clientes de beans de sesión interactúan a través de alguna de las interfaces declaradas (local o remota) y nunca directamente con el bean.
- Existen otras interacciones entre los EJB's y el contenedor (servidor):
  - ✓ Persistencia de beans,
  - ✓ Crear el mapeo entre los beans y la base.
  - ✓ Generación de código de las interfaces locales, remotas, etc.



Clases e Interfaces

Nomenclatura de nombres

- Cuando se habla de un proceso que tendrá su correspondiente EJB, por ejemplo Travel Agent, diremos **EJB Travel Agent**.
- Para referirse a los Message-driven Beans usamos el acrónimo **MDB**.
- Distinguimos entre los tipos de interfaz:
  - ✓ Remotas: `travelAgentRemote`.
  - ✓ Local: `travelAgentLocal`.
  - ✓ WebServices: `travelAgentWS`.
- Para la clase del bean, decimos simplemente **TravelAgent Bean**.

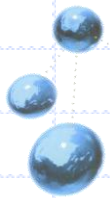


Clases e Interfaces

Interfaz Remota

```
import javax.ejb.Remote;

@Remote
public interface CalculatorRemote {
    public int add(int x, int y);
    public int subtract(int x, int y);
}
```

Anotación para indicar que esta clase define una interfaz remota.

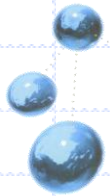


Clases e Interfaces

La clase bean

```
import javax.ejb.*;

@Stateless
public class CalculatorBean implements CalculatorRemote
   {
     public int add(int x, int y) {
         return x + y;
     }
     public int subtract(int x, int y) {
         return x - y;
     }
}
```

Anotación para indicar que esta clase es un EJB Session de tipo Stateless.



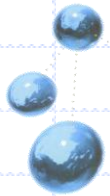
Anotaciones y Deployment descriptors

- El contenedor EJB maneja la seguridad, transacciones, y otros servicios primarios a través de información que determina a partir de anotaciones, y/o descriptores de despliegue XML.
- La especificación de las definiciones de servicios primarios tiene valores por defecto comunes que evitan en general que el programador agregue anotaciones o escriba descriptores de despliegue XML. Por ejemplo:
  - ✓ El default para transacciones es **REQUIRED**.
  - ✓ El default de semánticas de seguridad es **UNCHECKED**.

El Componente empresarial Bean

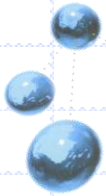
Archivos JAR

- Los archivos JAR almacenan:
  - ✓ Las clases bean de los EJB's.
  - ✓ Otros recursos para el funcionamiento de la aplicación: Aplicaciones Java, JavaBeans, Aplicaciones web y enterprise JavaBeans.
  - ✓ Puede contener el descriptor de despliegue si algunas anotaciones se hicieron en el XML.
- Cuando el contenedor abre el archivo JAR:
  - ✓ Busca por las clases anotadas como EJB's y/o lee el descriptor de despliegue.
  - ✓ Determina la manera en que se manejará la seguridad, transacciones, persistencia, etc.



Usando Enterprise & Entity Beans

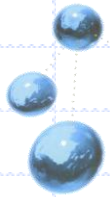
Session Beans



Session Beans

Modelando procesos de negocio

- Session Beans actuan como agentes que administran procesos o tareas de negocio. Estos son el lugar apropiado para agregar lógica de negocio.
- Los beans de sesión no son persistentes y nada en ellos se mapea a la BDD o es almacenado entre sesiones.
- Session beans trabajan con Entity Beans, datas y otros recursos para administrar workflows.



Session Beans

Proceso: reserva de cruceros

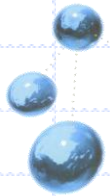
```
String creditCard = textField1.getText( );
int cabinID = Integer.parseInt(textField2.getText( ));
int cruiseID = Integer.parseInt(textField3.getText( ));

Customer customer = new Customer(name, address, phone);

TravelAgentRemote travelAgent = ...;
travelAgent.setCustomer(customer);

travelAgent.setCabinID(cabinID);
travelAgent.setCruiseID(cruiseID);

Reservation res = travelAgent.bookPassage(creditCard, price);
```

Se obtiene el número de tarjeta de crédito desde el campo de texto.

Crear una nueva sesión de TravelAGent, pasando una referencia a un Entity Bean cliente.

Se setean los ID's de la cabina y del crusero.

Usando el número de tarjeta y precio para reservar un pasaje.
Este método retorna un objeto de tipo Reservation



Session Beans

Proceso: Travel Agent Session Bean

```
@Stateful
public class TravelAgentBean implements TravelAgentRemote {
    @PersistenceContext private EntityManager entityManager;
    @EJB private ProcessPaymentRemote process;

    private Customer customer;
    private Cruise cruise;
    private Cabin cabin;

    public void setCustomer(Customer cust) {
        entityManager.create(cust);
        customer = cust;
    }
    public void setCabinID(int id) {
       cabin = entityManager.find(Cabin.class, id);
    }
    public void setCruiseID(int id) {
       cruise = entityManager.find(Cruise.class, id);
    }

    ...
```

Se obtiene una referencia al Entity Manager.

Se obtiene una referencia al Bean de sesión ProcessPayment.

Session Beans

Proceso: Travel Agent Session Bean

```
public Reservation bookPassage(String card, double price)
    throws IncompleteConversationalState {
    if (customer == null ||cruise == null ||cabin == null){
        throw new IncompleteConversationalState( );
    }
    try {

        Reservation reservation =
            new Reservation(customer,cruise,cabin,price,new Date( ));

        entityManager.persist(reservation);

        process.byCredit(customer,card,price);


        return reservation;
    }catch(Exception e){
        throw new EJBException(e);
    } } }
```

Se crea un objeto Reseruación

Se persiste en la BDD.

Se invoca un proceso de negocio.

Session Beans

Statefull & Stateless

Statefull

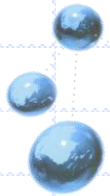
- Mantiene un estado conversacional con un cliente.
- El estado conversacional no es persistido, sino mantenido en memoria.
- Una vez que el cliente libera o termina el TravelAgent EJB, el estado conversacional se pierde para siempre.

Stateless

- No mantienen ningún estado conversacional.
- Cada método es independiente de los otros y utiliza sólo la información que se le pasa como parámetro.
- El ProcessPayment EJB es un buen ejemplo de un stateless: no mantiene un estado conversacional entre una invocación y la siguiente.

Message Driven Beans

- Los MDB son puntos de integración para otras aplicaciones que necesitan trabajar con aplicaciones EJBs vía JMS.
- MDB pueden soportar cualquier sistema de mensajería que implemente el contrato JCA 1.5 (Java Connector Architecture) correcto.
- Sin embargo soporte para JMS-MDB en EJB 3.0 es obligatorio. Nos enfocaremos en este tipo.
- En varios sentidos, un MDB juega el mismo rol que un EJB Sesión Stateless.
- Un MDB responde a un mensaje a través de su método `onMessage()`.
- Cómo los mensajes son asíncronos, el cliente que envía el mensaje no espera una respuesta.



Message Driven Beans

Ejemplo Travel Agent MDB

```
@MessageDriven
public class ReservationProcessorBean implements javax.jms.MessageListener {
    @PersistenceContext private EntityManager entityManager;
    @EJB private ProcessPaymentRemote process;

    public void onMessage(Message message) {
        try {
            MapMessage reservationMsg = (MapMessage) message;

            Customer customer = (Customer)reservationMsg.getObject("Customer");
            int cruisePk = reservationMsg.getInt("CruiseID");
            int cabinPk = reservationMsg. getInt("CabinID");
            double price = reservationMsg.getDouble("Price");

            String card = reservationMsg.getString("card");
            entityManager.persist(customer);
```

Es un conjunto de pares de nombre-valor.

Message Driven Beans

Ejemplo Travel Agent MDB

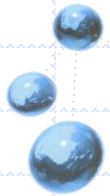
```
            Cruise cruise = entityManager.find(Cruise.class, cruisePK);
            Cabin cabin = entityManager.find(Cabin.class, cabinPK);

            Reservation reservation =
                new Reservation(customer,cruise,cabin,price,new Date( ));

            entityManager.create(reservation);

            process.byCredit(customer,card,price);

        } catch(Exception e) {
            throw new EJBException(e);
        }
    }
}
```

El contrato del Bean Container

El Contenedor EJB

Responsabilidades

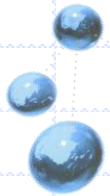
- El **Contenedor EJB** actúa como un intermediario entre el bean y el servidor EJB.
- La diferencia entre **Contenedor EJB** y servidor no está bien clara, sin embargo, la especificación EJB está definida en términos de las responsabilidades del contenedor.
- Responsabilidades:
  - ✓ Manejar el ciclo de vida de los EJB.
  - ✓ Proveerle acceso a recursos.
  - ✓ Proveer servicios primarios en tiempo de ejecución:
    - Transacciones
    - Seguridad
    - Concurrencia
    - Nombres



El Contenedor EJB

Interacción

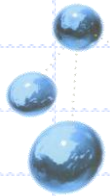
- Los EJB's intearctúan con el contenedor a través de una intefaz bien definida.
- Los EJB se suscriben a varios eventos del ciclo de vida que el contenedor emite.
  - ✓ El EJB se registra a estos eventos a través de anotaciones hechas sobre sus métodos.
  - ✓ En tiempo de ejecución el contenedor invoca estos métodos cuando ocurren los eventos.
- Ejemplo.
  - ✓ El servidor invocará los métodos de un EJB que estén anotados con la etiqueta `@javax.annotation.PostConstruct`, cuando reserve espacio e inyecte referencias a los servicios que utilizará una instancia.



El Contenedor EJB

El Contexto de un Contenedor

- La interfaz `javax.ejb.EJBContext` es parte del contrato del Contenedor EJB y debe ser implementada por el Contenedor.
  - ✓ EJB's de sesión utilizan la subclase `javax.ejb.SessionContext`.
  - ✓ MDB utilizan la subclase `javax.ejb.MessageDriven-Context`.
- Estas clases le proveen al EJB de información sobre el contexto en el que se encuentra: contenedor, el cliente que usa el EJB, y el EJB en sí mismo.
- El contrato con el contenedor también incluye un **JNDI**, llamado **Environment-naming context**, utilizado para buscar recursos que el EJB necesita.



Preguntas?