

# Sesiones Enterprise JavaBeans

# Introducción

## EJB ?

- Componente (clase java) del servidor.
- Encapsulan la lógica del negocio.
- Posee una pila integrada para:
  - Acceso remoto,
  - Mensajería,
  - Servicio web (SOAP, REST), etc.

# Introducción

## EJB ?

- **Modelo** muy simple (hoy en día) para desarrollar aplicaciones **distribuidas, transaccionales, seguras y portátiles.**
- Un simple POJO (Plain Old Java Object) **mas** anotaciones, podrá desplegarse en un **contenedor EJB.**

# Modelo del componente EJB

EJB define tres tipos:

- *Session beans*: Operan en contextos transaccionales y distribuidos.
- *Message-driven beans* (MDBs): responde a eventos externos.

Complementariamente, JPA define:

- *Entidades*: clases que representan tablas de una bd.

# Contenedor EJB

- Es el entorno donde operan los componentes EJB.
- Ofrece servicios:
  - de transacción y de seguridad,
  - pool de recursos y almacenamiento,
  - soporte de concurrencia, etc,

# Sesiones Beans

- El nombre *sesión* implica que una instancia del bean está disponible durante una "unidad de trabajo".
- Una sesión Bean Enterprise es invocada por un cliente con el fin de realizar una operación de negocio específica.

[Especificación EJB 3.1](#)

# Tipos de sesiones Beans

- Existen tres tipos de sesiones:
  1. **Stateful - Con estado:** El estado del bean se mantiene a través de varias llamadas a métodos.

El "estado" se refiere a los valores de sus variables de instancia. Debido a que el cliente interactúa con el bean, este estado se llama a menudo **el estado de conversación.**

# Tipos de sesiones Beans

2. **Stateless:** Los beans sin estado se utilizan para las operaciones que se pueden producir en una sola llamada al método. Cuando el método termina el procesamiento, no se conserva el estado del bean específico del cliente.

# Tipos de sesiones Beans

3. **Singleton:** Un bean de sesión singleton se instancia una vez por aplicación, y existe para el ciclo de vida de la aplicación.  
Esta sesión está diseñada para ser compartida y accedida concurrentemente por los clientes

## Sesion Bean JPA

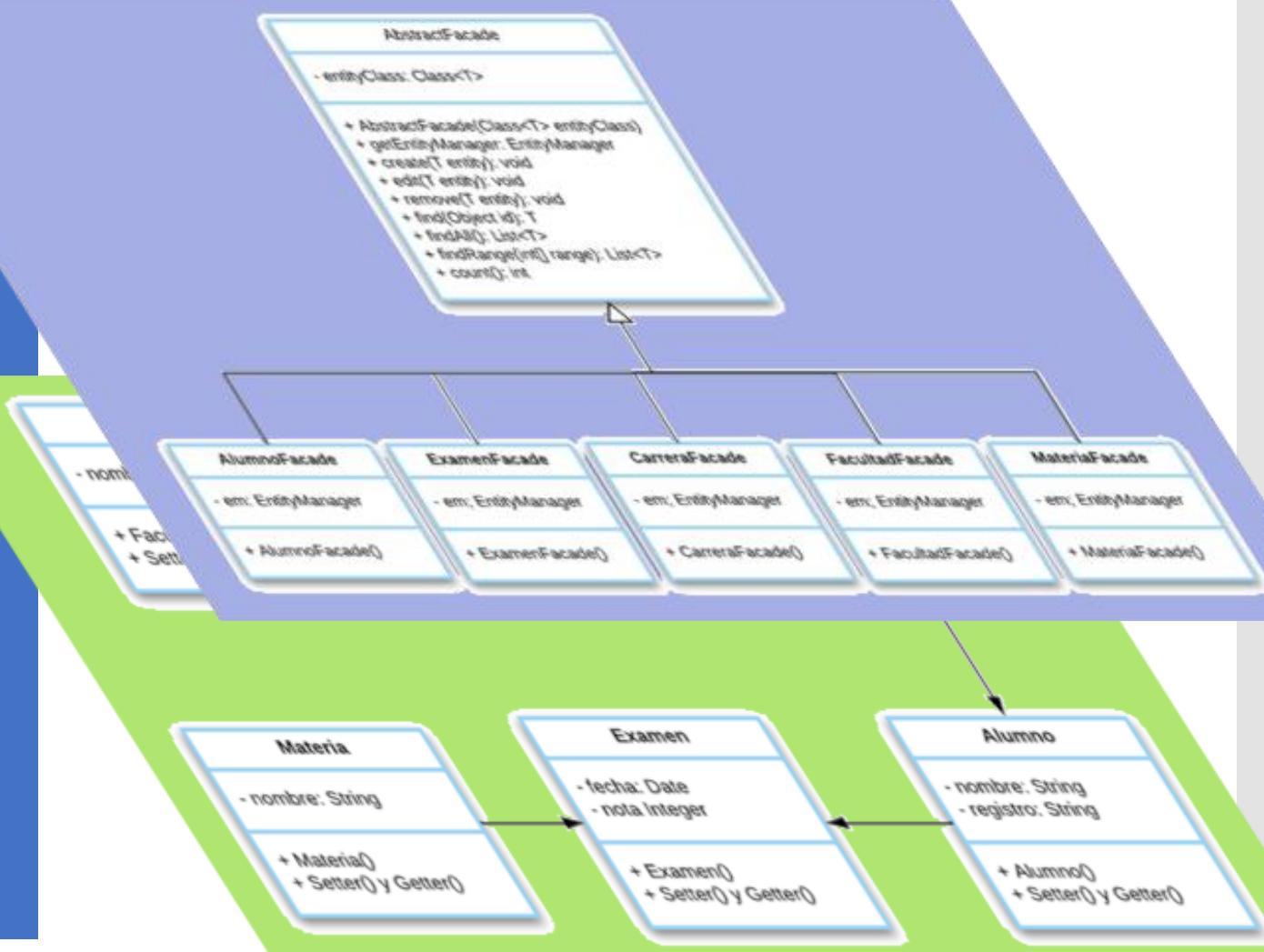
- Los beans de sesión proporcionan servicios (transacciones, seguridad, control de acceso) a una aplicación cliente.
- Las entidades representan los datos de negocio.



## *Patrón Facade*

Un bean de sesión proporciona una interfaz conveniente para la manipulación de las entidades (crear, recuperar, actualizar y eliminar) –  
CRUD -

# Aplicación



# Conclusión

Las tecnologías vistas funcionan de manera integrada para configurar rápidamente el modelo de cada aplicación, conservando las pautas propuestas por el patrón [MVC](#).

# App Java Enterprise

## Incluye:

- Proyecto de una Aplicación Web
- Diseño de la interfaz de usuario
- Modelo de Datos
- **Conexión a MySQL**
- Generación de Entidades
- Generación de Sesiones Beans
- Diseño con patrones, Model-View-Controller (MVC) y Session Facade

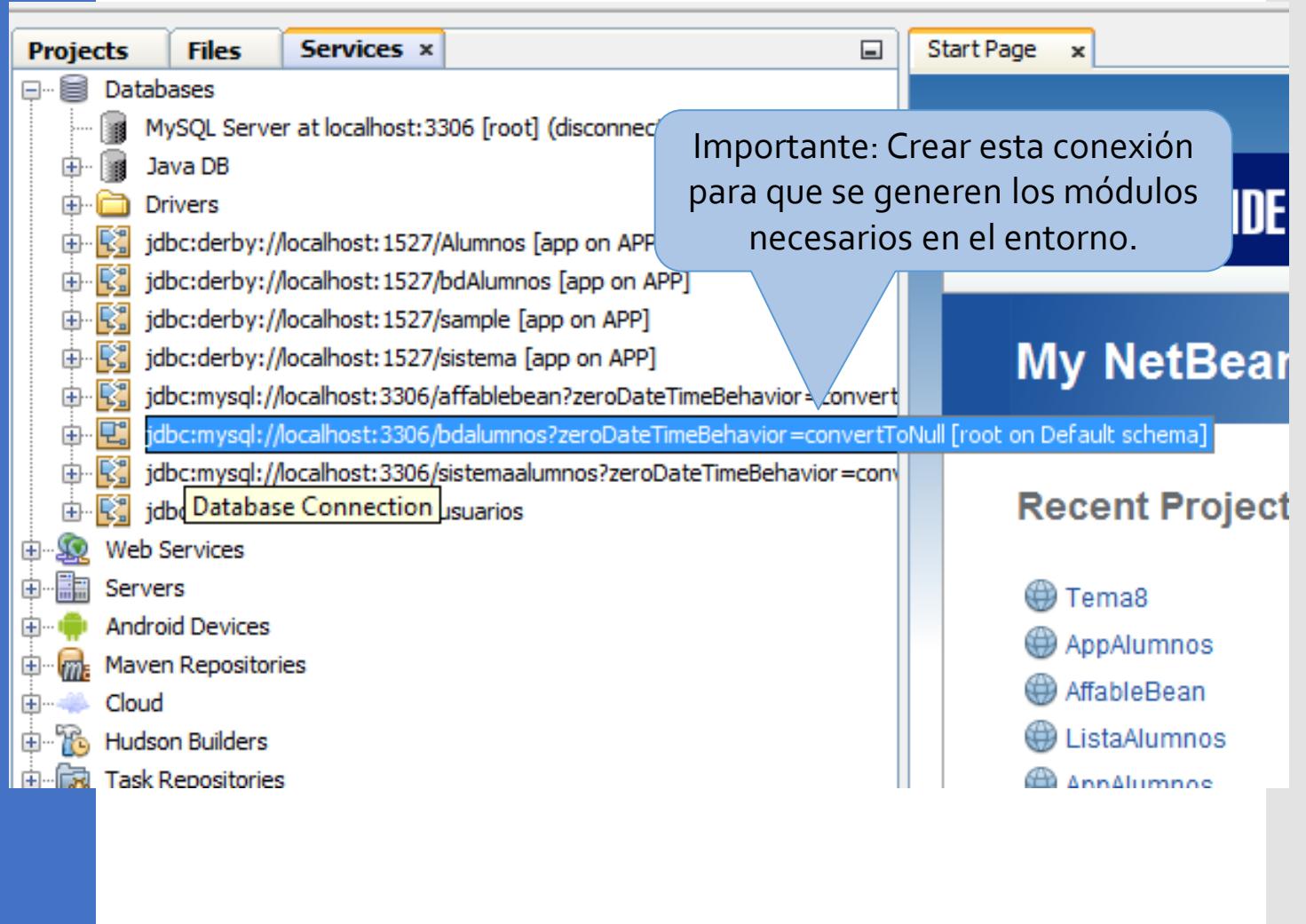
# Conexión a MySql sobre GlassFish

Esta comunicación se hace posible con la API de Java Database Connectivity (**JDBC**). Es una biblioteca de integración **contenida en el JDK**, requerida para comunicar SQL y Java.

Se crea una **pool** de conexiones en el servidor GlassFish. Para que el servidor se comunique directamente con la base de datos, se requiere convertir las llamadas JDBC directamente en un protocolo específico de MySQL.

Un **data source** permite a las aplicaciones la conexión a una bd, que se obtiene del pool de conexiones, buscando en el directorio de nombres (JNDI).

# Crear la conexión jdbc:mysql



# Crear conexión Pool y DataSource

Existen dos caminos:

1. **Consola de GlassFish:** se introducen manualmente los detalles de conexión de bd (usuario, contraseña y URL).
2. **IDE de NetBeans:** extrae detalles de conexión directamente de la conexión bd existente, eliminando así posibles problemas de conectividad.

<https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-usagenotes-glassfish-config.html>

<https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-api-changes.html>

# 1. Consola de GlassFish

1. Desde el IDE iniciar el servidor GlassFish.
2. Abrir **View Domain Admin Console**  
[ user: admin pass: admadmin ]
3. Expandimos Resources > JDBC y click **JDBC Conection pools**
4. Click **NEW** y completamos:  
**Name:** BDAlumnosPool  
**Resource Type:** javax.sql.ConnectionPoolDataSource  
**Database Vendor:** MySql      **Next**
5. **Finish**

6. Click en el pool creado.

7. Cambiar a:

**Datasource Classname:**

MySQL 5: com.mysql.jdbc.jdbc2.optional.MysqlDataSource

MySQL 8: com.mysql.cj.jdbc.MysqlDataSource

8. **Save**

9. Click en tab **Additional Properties** y agregar las propiedades:

**user: root**

**password: .....**

**useSSL: false**

**url:**

**bdalumnos?zeroDateTimeBehavior=ConvertToNull&serverTimezone=UTC**

Nota: Prestar atención al **url** y respetar mayúsculas

10. **Save**

11. Click TAB **General** y luego click Ping. **Deberias ver msg Ping Exitoso**

# Ejemplo de propiedades

The screenshot shows the Eclipse GlassFish Admin Console interface. At the top, there are links for 'Home' and 'About...', and status information: 'User: admin | Domain: domain1 | Server: localhost'. Below this is the title 'Eclipse GlassFish'. A button 'Enable logging commands' is visible. The main content area has tabs for 'General', 'Advanced', and 'Additional Properties', with 'Additional Properties' currently selected. The section title is 'Edit JDBC Connection Pool Properties' with the subtitle 'Modify properties of an existing JDBC connection pool.' A note states 'Pool Name: BDAlumnosPool'. Below this is a table titled 'Additional Properties (8)' with columns 'Select', 'Name', and 'Value'. The table lists the following properties:

Select	Name	Value
<input type="checkbox"/>	password	root
<input type="checkbox"/>	databaseName	bdalumno
<input type="checkbox"/>	serverName	localhost
<input type="checkbox"/>	user	root
<input type="checkbox"/>	portNumber	3306
<input type="checkbox"/>	useSSL	false
<input type="checkbox"/>	url	jdbc:mysql://localhost:3306/bdalumno?zeroDa
<input type="checkbox"/>	allowPublicKeyRetrieval	true

General Advanced Additional Properties

Ping Succeeded

## Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

[Load Defaults](#) [Flush](#) [Ping](#)

### General Settings

**Pool Name:** BDAlumnosPool

**Resource Type:** `javax.sql.ConnectionPoolDataSource` ▾

Must be specified if the datasource class implements more than 1 of the interface.

**Datasource Classname:** `com.mysql.cj.jdbc.MysqlDataSource`

Vendor-specific classname that implements the DataSource and/or XADatasource APIs

**Driver Classname:**

Vendor-specific classname that implements the `java.sql.Driver` interface.

**Ping:**

When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attrib

**Deployment Order:**

`100`

Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

**Description:**

12. Click **JDBC Resources**
13. Click **NEW** y completamos:  
**JNDI Name:** jdbc/bdalumnos  
**Connection Pool:** BDAlumnosPool
14. **OK**

# Otra forma

## 2. IDE de Netbeans

1. Crear un nuevo archivo en el menu principal del IDE.
2. Elegir categoria **GlassFish** y luego seleccionar **JDBC Resource**
3. Seguir el asistente:
  1. Tildar **Create New JDBC Connection Pool**
  2. Setear el data source:

JNDI Name: `jdbc/bdalumnos` // POR CONVENCION DE  
NOMBRE JNDI

Object Type: `user`

Enabled: `true` //Next
4. //Next
5. Elegir nombre del pool **BDAAlumnosPool**.

Tildar **Extract from Existing Connection**, y elegir  
`jdbc:mysql://localhost:3306/bdalumnos`. //Next
6. Seleccionar:

Datasource Classname:

MySQL 5: `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`  
MySQL 8: `com.mysql.cj.jdbc.MysqlDataSource`

Resource Type: `javax.sql.ConnectionPoolDataSource`

User: ....

Password:... //Finish
6. Revisar el archivo `glassfish-resources.xml` de tu proyecto.

# App Java Enterprise

## Incluye:

- Proyecto de una Aplicación Web
- Diseño de la interfaz de usuario
- Modelo de Datos
- Conexión a MySQL
- **Generación de Entidades**
- Generación de Sesiones Beans
- Diseño con patrones, Model-View-Controller (MVC) y Session Facade

# Generar Entidades

New Entity Classes from Database

**Steps**

1. Choose File Type
- 2. Database Tables**
3. Entity Classes
4. Mapping Options

**Database Tables**

Data Source:

Available Tables:

- aplicacionDS
- java:comp/DefaultDataSource
- jdbc/\_default
- jdbc/\_TimerPool
- jdbc/bdalumnos
- jdbc/sample
- tema8DS

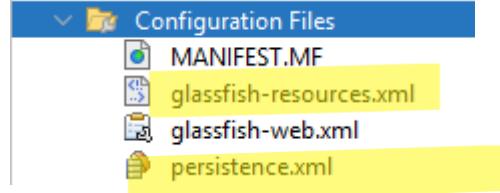
< Remove | Add All >> | << Remove All | Any |  Include Related Tables

**!** Select the table source.

< Back | Next > | Finish | Cancel | Help

# Primera parada

- IMPORTANTE: Si existen, **borrar** del proyecto:



- Generar las conexiones necesarias para crear las entidades de la base de datos alumnos.



# Entendiendo Entidades

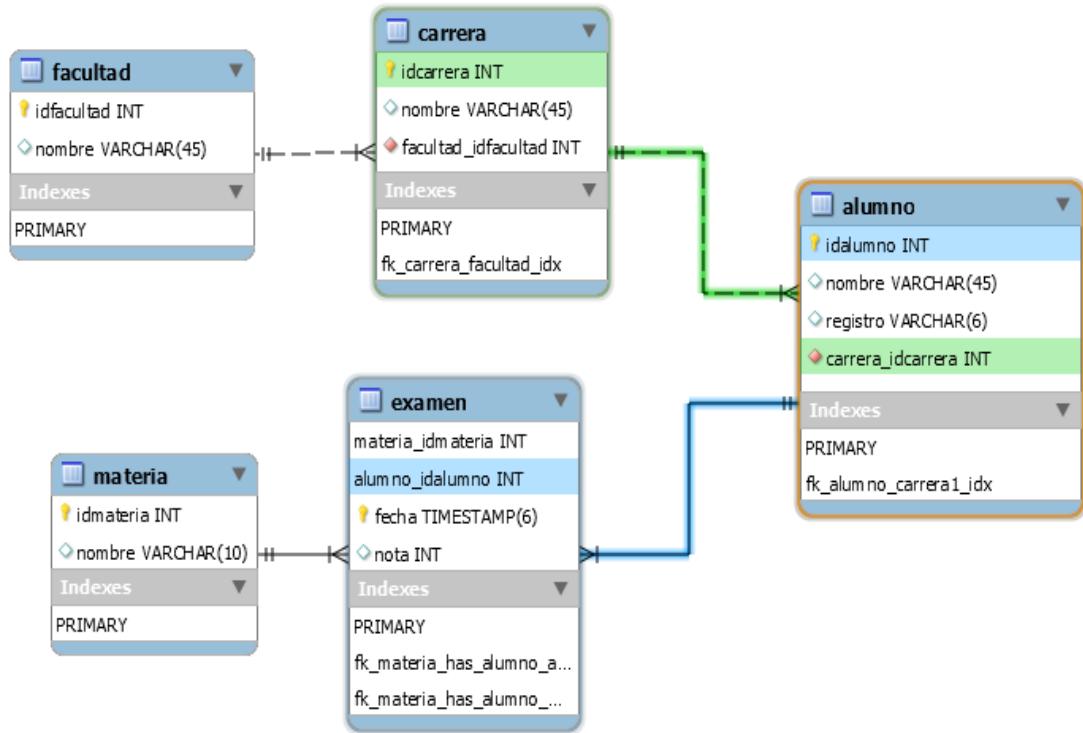
- Es necesario analizar como se mapea el modelo entidad-relación con el modelo OO:
  - JPA realiza el mapeo por [configuración por anotaciones](#).
  - La cardinalidad puede ser: uno-a-uno, uno-a-muchos, muchos-a-uno o muchos-a-muchos.

# Mapeo de Relaciones

- En el mundo de la POO una **asociación** liga objetos de una clase a objetos de otra:
  - Varios **tipos** de asociaciones pueden existir entre clases.
  - Tiene **dirección**: unidireccional o bidireccional.
  - Tiene **multiplicidad** o cardinalidad.

# ...Mapeo de Relaciones

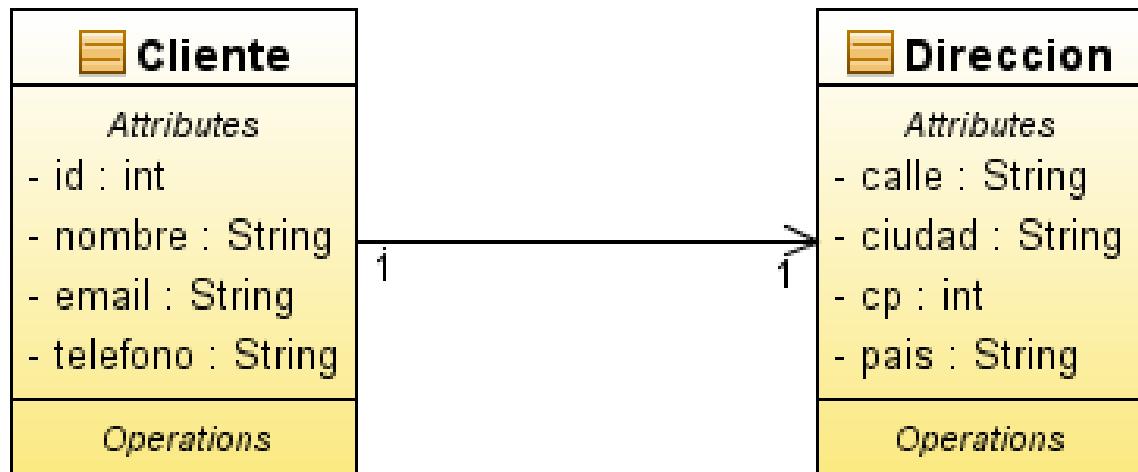
- En el mundo de BD relacional, una base de datos es una colección de tablas:
  - Las tablas se relacionan a través de referencias. Estas pueden ser modeladas como:
    - Join de columnas: usando claves primarias.
    - Join de tablas.
  - Toda columna que referencia a una clave primaria de otra tabla se la llama clave foránea.



## ...Mapeo de Relaciones

- JPA usa **configuración por anotación** para mapear las asociaciones.
- Para personalizar el mapeo se deben especificar distintas anotaciones.
- La cardinalidad puede ser: uno-a-uno, uno-a-muchos, muchos-a-uno o muchos-a-muchos.

# OOP: Relación uno-a-uno (UniDirac)



...Relación  
uno-a-uno  
(UniDir)

**@Entity**

public class Cliente {

**@Id @GeneratedValue**

private Long id;

private String nombre;

private String email;

private String telefono;

Configuración por anotaciones:

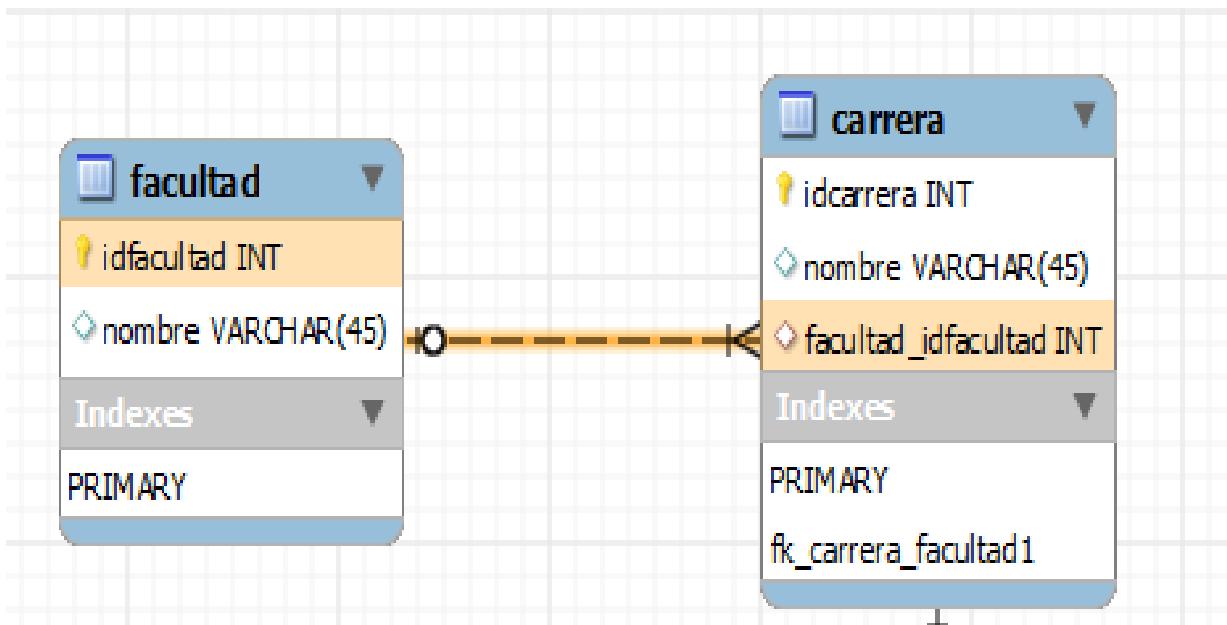
**@OneToOne ( fetch = FetchType.LAZY )**

**@JoinColumn ( name = "foranea", nullable = false )**

**private Direccion direccion;**

// Constructores, getters, setters}

## Relación uno a muchos



```
@Entity
@Table(name = "facultad")
public class Facultad implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idfacultad")
    private Integer idfacultad;
    @Size(max = 45)
    @Column(name = "nombre")
    private String nombre;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "facultadIdfacultad")
    private Collection<Carrera> carreraCollection;
```

```
public class Carrera implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idcarrera")
    private Integer idcarrera;
    @Size(max = 45)
    @Column(name = "nombre")
    private String nombre;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "carreraIdcarrera")
    private Collection<Alumno> alumnoCollection;
    @JoinColumn(name = "facultad_idfacultad", referencedColumnName = "idfacultad")
    @ManyToOne(optional = false)
    private Facultad facultadIdfacultad;
```

# Mapeo de Herencia

- La herencia es un mecanismo de programación que permite **reusar** código.
- Es un concepto completamente desconocido en el mundo relacional.
- El mapeo a una BD relacional no es directo.

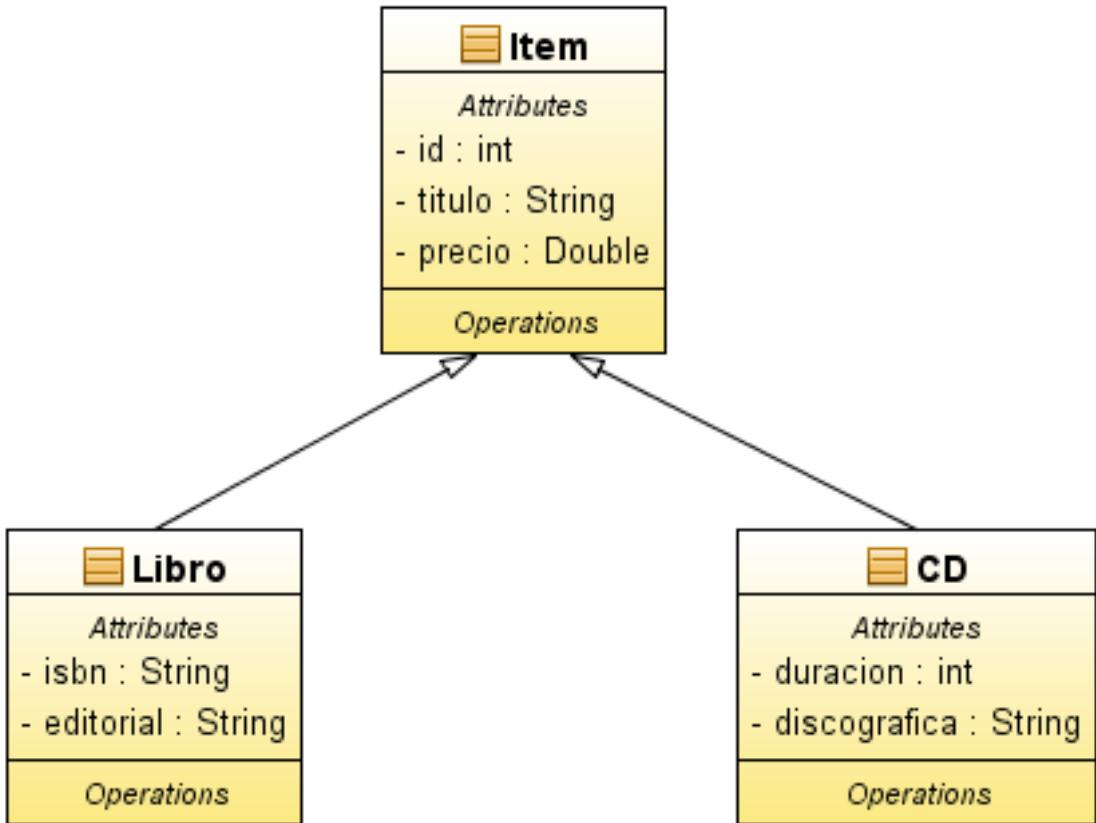
## ...Mapeo de Herencia

### JPA tiene tres estrategias:

1. **Una simple tabla por jerarquía:** Todos los atributos de una jerarquía de entidades son aplanados en una sola tabla.
2. **Joined subClases:** Cada entidad, concreta o abstracta, es mapeada en su propia tabla.
3. **Una tabla por clase concreta:** Cada entidad concreta es mapeada en tabla por separado.

Nota: La tercera estrategia esta a prueba en la versión JPA 2.0 y por lo tanto debería evitarse su uso.

# ...Mapeo de Herencia



## ...Mapeo de Herencia

1. Estrategia: Una simple tabla por jerarquía.

ITEM		
+ID	bigint	Nullable = false
<b>DTYPE</b>	<b>varchar(31)</b>	<b>Nullable = true</b>
TITULO	varchar	Nullable = true
PRECIO	double	Nullable = true
ISBN	varchar	Nullable = true
EDITORIAL	varchar	Nullable = true
DURACION	integer	Nullable = true
DISCOGRAFICA	varchar	Nullable = true

Es la estrategia por defecto y por lo tanto **no se usa @Inheritance**

## ...Mapeo de Herencia

### 2. Estrategia: Joined subClases.

```
@Entity  
@Inheritance(strategy = InheritanceType.JOINED)  
public class Item {  
    @Id @GeneratedValue  
    private Long id;  
    private String titulo;  
    private Double precio;  
  
    // Constructores, getters, setters }
```

## LIBRO

+# ID	bigint	Nullable = false
ISBN	varchar	Nullable = true
EDITORIAL	varchar	Nullable = true

## CD

+# ID	bigint	Nullable = false
DURACION	integer	Nullable = true
DICOGRAFICA	varchar	Nullable = true

## ITEMS

+# ID	bigint	Nullable = false
DTYPE	varchar(31)	Nullable = true
TITULO	varchar	Nullable = true
PRECIO	double	Nullable = true

## ...Mapeo de Herencia

3. Estrategia: Una tabla por clase concreta

```
@Entity  
@Inheritance(strategy =  
    InheritanceType.TABLE_PER_CLASS)  
public class Item {  
    @Id @GeneratedValue  
    private Long id;  
    private String titulo;  
    private Double precio;  
  
    // Constructores, getters, setters }
```

## LIBRO

+ ID	bigint	Nullable = false
TITULO	varchar	Nullable = true
PRECIO	double	Nullable = true
ISBN	varchar	Nullable = true
EDITORIAL	varchar	Nullable = true

## CD

+ ID	bigint	Nullable = false
TITULO	varchar	Nullable = true
PRECIO	double	Nullable = true
DURACION	integer	Nullable = true
DICOGRAFICA	varchar	Nullable = true

## ITEMS

+ ID	bigint	Nullable = false
TITULO	varchar	Nullable = true
PRECIO	double	Nullable = true

# App Java Enterprise

## Incluye:

- Proyecto de una Aplicación Web
- Diseño de la interfaz de usuario
- Modelo de Datos
- Conexión a MySQL
- Generación de Entidades
- **Generación de Sesiones Beans**
- Diseño con patrones, Model-View-Controller (MVC) y Session Facade

# Agregar sesiones

El asistente genera una “[fachada](#)” sesión EJB (patrón Facade) para cada una de las clases de entidad que se han creado. Cada sesión bean contendrá métodos de acceso básicos para su respectiva clase de entidad.

# Sesión Facade

## Características:

- Es un **patrón de diseño** que, intenta resolver los problemas comunes que surgen en un entorno de aplicaciones de varios niveles, tales como:
  - **Acoplamiento** lo que conduce a la dependencia entre los clientes y los objetos de negocio.
  - Demasiadas llamadas a métodos entre el cliente y el servidor, lo que lleva a **problemas de rendimiento** de la red
  - La **falta de una estrategia** uniforme de acceso de cliente, exponiendo los objetos de negocio a un mal uso

# Sesión Facade

Características:

- **Abstira las interacciones** de objetos de negocio.  
Proporciona una capa de servicio que expone sólo la funcionalidad requerida.

Se crea un  
nuevo  
componente

New File

Steps

1. Choose File Type  
2. ...

Choose File Type

Project: AppAlumnos

Filter:

Categories:

- JavaServer Faces
- Bean Validation
- Spring Framework
- Enterprise JavaBeans
- Contexts and Dependency Inj
- Java
- Swing GUI Forms
- JavaBeans Objects
- AWT GUI Forms

File Types:

- Session Bean
- Timer Session Bean
- Message-Driven Bean
- Session Beans For Entity Classes
- Standard Deployment Descriptor

Description:

Creates Session EJB Facade based on Entity class. This template creates Session EJB for every Entity class with basic access methods.

< Back    Next >    Finish    Cancel    Help

- Se eligen todas las entidades

New Session Beans For Entity Classes

Steps

1. Choose File Type
2. **Entity Classes**
3. Generated Session Beans

Entity Classes

Available Entity Classes:

entidades.Alumno  
entidades.Carrera  
entidades.Facultad  
entidades.Materia

Add >  
< Remove  
Add All >>  
<< Remove All

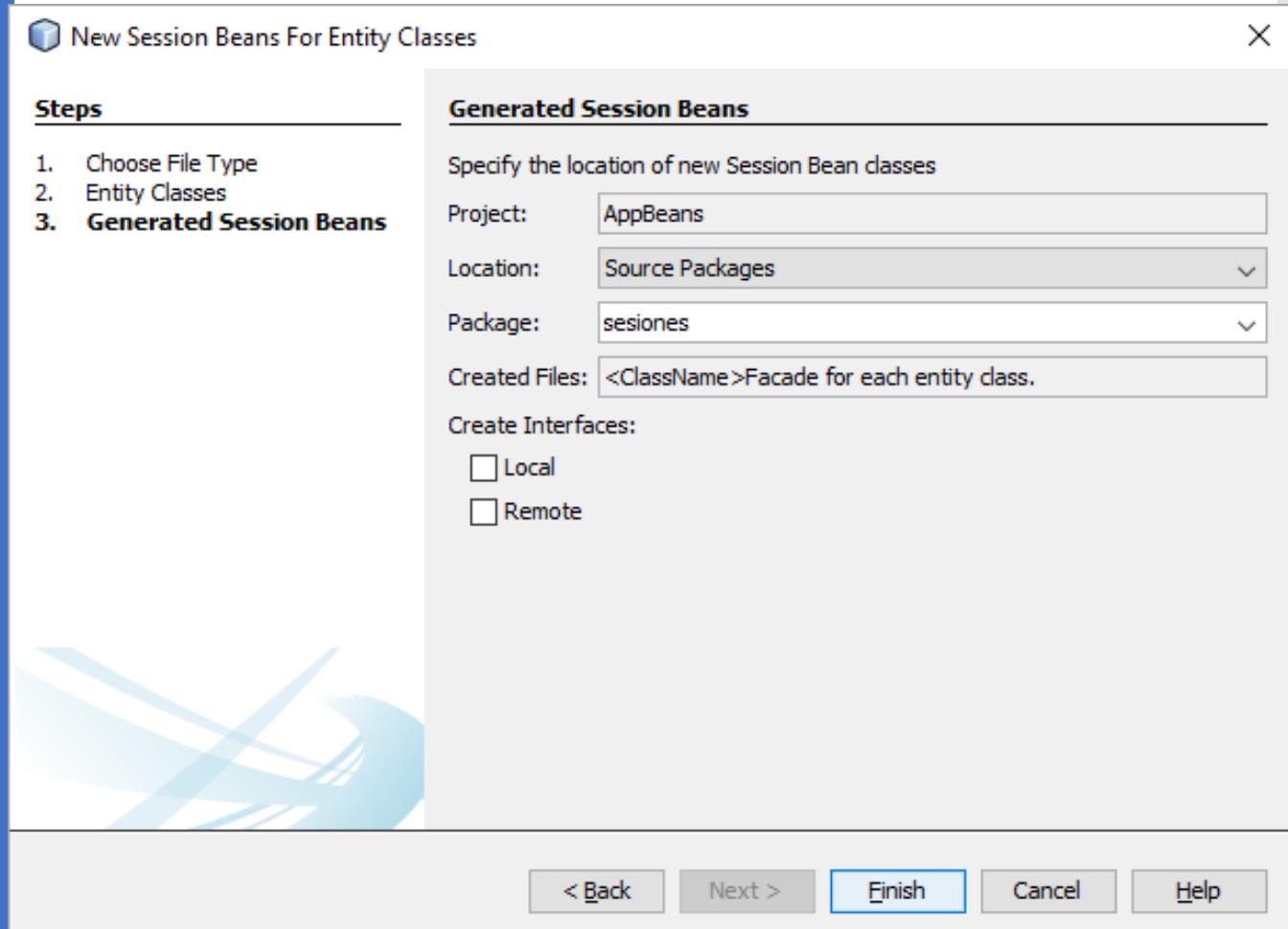
Selected Entity Classes:

Include Referenced Classes

! Select entity classes.

< Back Next > Finish Cancel Help

- Se  
ingresa el  
paquete



The screenshot shows a Java development environment with the following interface elements:

- Project Explorer (Left):** Shows the project structure under "AppAlumnos".
  - Source Packages:
    - controlador (containing Manejador.java)
    - entidad
    - sesion (containing AbstractFacade.java)
  - Test Packages
  - Libraries
  - Test Libraries
  - Configuration Files:
    - MANIFEST.MF
    - glassfish-web.xml
    - persistence.xml
- Editor (Center):** Displays the code for `AbstractFacade.java`. The code is annotated with ellipses and line numbers. A yellow highlight covers the entire class definition from the opening brace to the closing brace.
- Navigator (Bottom Left):** Shows the members of the current file, which is currently empty.

```
1  ...4 lines
5  package sesion;
6
7  import jakarta.persistence.EntityManager;
8  import java.util.List;
9
10  /**...5 lines */
11  public abstract class AbstractFacade<T> {
12
13      private Class<T> entityClass;
14
15      public AbstractFacade(Class<T> entityClass) {...3 lines}
16
17      protected abstract EntityManager getEntityManager();
18
19      public void create(T entity) {...3 lines}
20
21      public void edit(T entity) {...3 lines}
22
23      public void remove(T entity) {...3 lines}
24
25      public T find(Object id) {...3 lines}
26
27      public List<T> findAll() {...5 lines}
28
29      public List<T> findRange(int[] range) {...8 lines}
```

AlumnoFacade.java

Source History

```
1  [+] ... 4 lines
5  package session;
6
7  [-] import entidad.Alumno;
8  [-] import jakarta.ejb.Stateless;
9  [-] import jakarta.persistence.EntityManager;
10 [-] import jakarta.persistence.PersistenceContext;
11
12 [-] /**
13  * 
14  * @author Usuario
15  */
16 @Stateless
17 public class AlumnoFacade extends AbstractFacade<Alumno> {
18
19     @PersistenceContext(unitName = "AppAlumnosPU")
20     private EntityManager em;
21
22     @Override
23     protected EntityManager getEntityManager() {
24         return em;
25     }
26
27     public AlumnoFacade() {
28         super(Alumno.class);
29     }
30 }
```

- `@PersistenceContext` se utiliza para injectar un EntityManager gestionada por contenedor en la clase.
- En este caso se basa en el contenedor EJB GlassFish para abrir y cerrar EntityManagers como y cuando sea necesario.
- El elemento `unitName` especifica la unidad de persistencia `AppBeansPU`, que ha sido definido en el archivo de persistence.xml de la aplicación.

```
public class Manejador extends HttpServlet {  
    //  
    @EJB  
    private AlumnoFacade alumnoF;
```

```
        case "/Listar":  
            /*           em = emf.createEntityManager();  
                         Query q = em.createNamedQuery("Alumno.findAll");  
                         List todos = q.getResultList();  
            */  
            request.setAttribute("lista", alumnoF.findAll());  
            //      getServletContext().setAttribute("lista", emf.findAll());  
            url = "/WEB-INF/vista/" + pathUsuario + ".jsp";  
            break;
```



ListarAlumnos.jsp

x

```
13 <html>
14     <head>
15         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
16         <title>JSP Page</title>
17     </head>
18     <body>
19         <h1>Listado de alumnos</h1><hr>
20         <table id="lista" border="3">
21             <tr>
22                 <th bgcolor="red">Registro</th>
23                 <th bgcolor="red">Nombre</th>
24                 <th bgcolor="red">Carrera</th>
25             </tr>
26             <c:forEach var="a" begin="0" items="${lista}">
27                 <tr>
28                     <td>${a.getRegistro()}&nbsp;&nbsp;</td>
29                     <td>${a.getNombre()}&nbsp;&nbsp;</td>
30                     <td>${(a.getCarraIdcarrera()).getNombre()}&nbsp;&nbsp;</td>
31             </tr>
32
33         </c:forEach>
34
35     </table>
36
37     </body>
38
39 </html>
```

## Segunda parada

1. Generar los Beans en tu aplicación.
2. Dejar a punto y desplegar.
3. Realizar los cambios necesarios para que aparezca en el listado de alumnos el nombre facultad.

