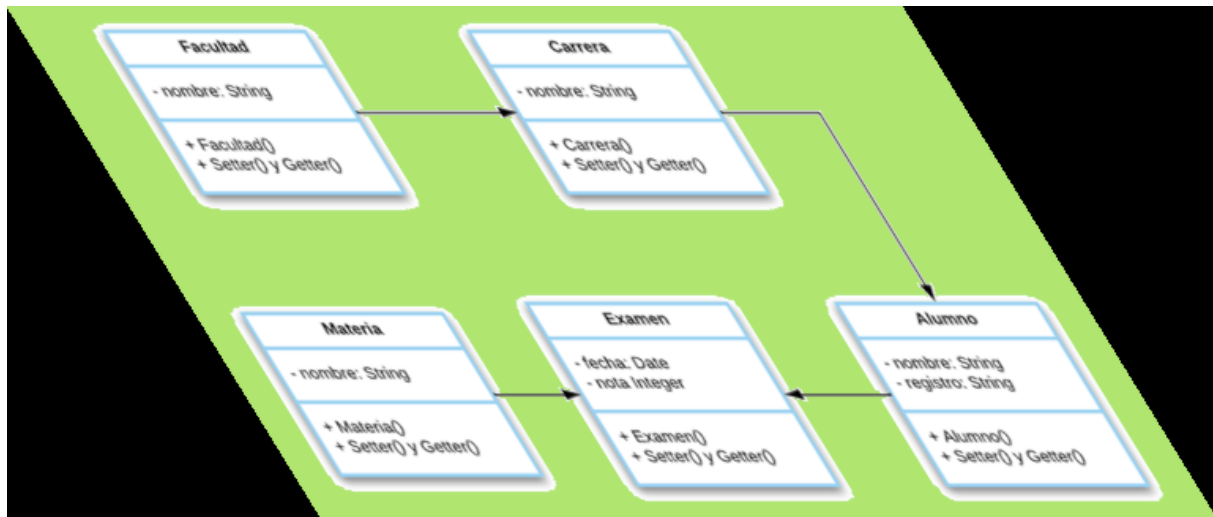


# Tutorial: Entidades JPA y Session Facades (EJB)

## Introducción

Este tutorial muestra paso a paso cómo generar entidades JPA y crear Session Facades (EJB stateless) para exponer operaciones CRUD básicas. Se IGNORA la configuración del servidor GlassFish y los pools de conexión, centrándonos en la generación de entidades, relaciones y la creación de los EJBs tipo facade.

Se incluyen ejemplos de código (Java) que podés copiar y adaptar a tu proyecto.



## 1. Generación de Entidades JPA

### 1) Modelado de la base de datos y entidades JPA

- Identificar tablas y relaciones (PK, FK, cardinalidades).
- Cada tabla suele mapearse a una clase anotada con `@Entity`.
- Definir el identificador con `@Id` y estrategia de generación `@GeneratedValue`.

Ejemplo de entidad básica:

```
import javax.persistence.*;

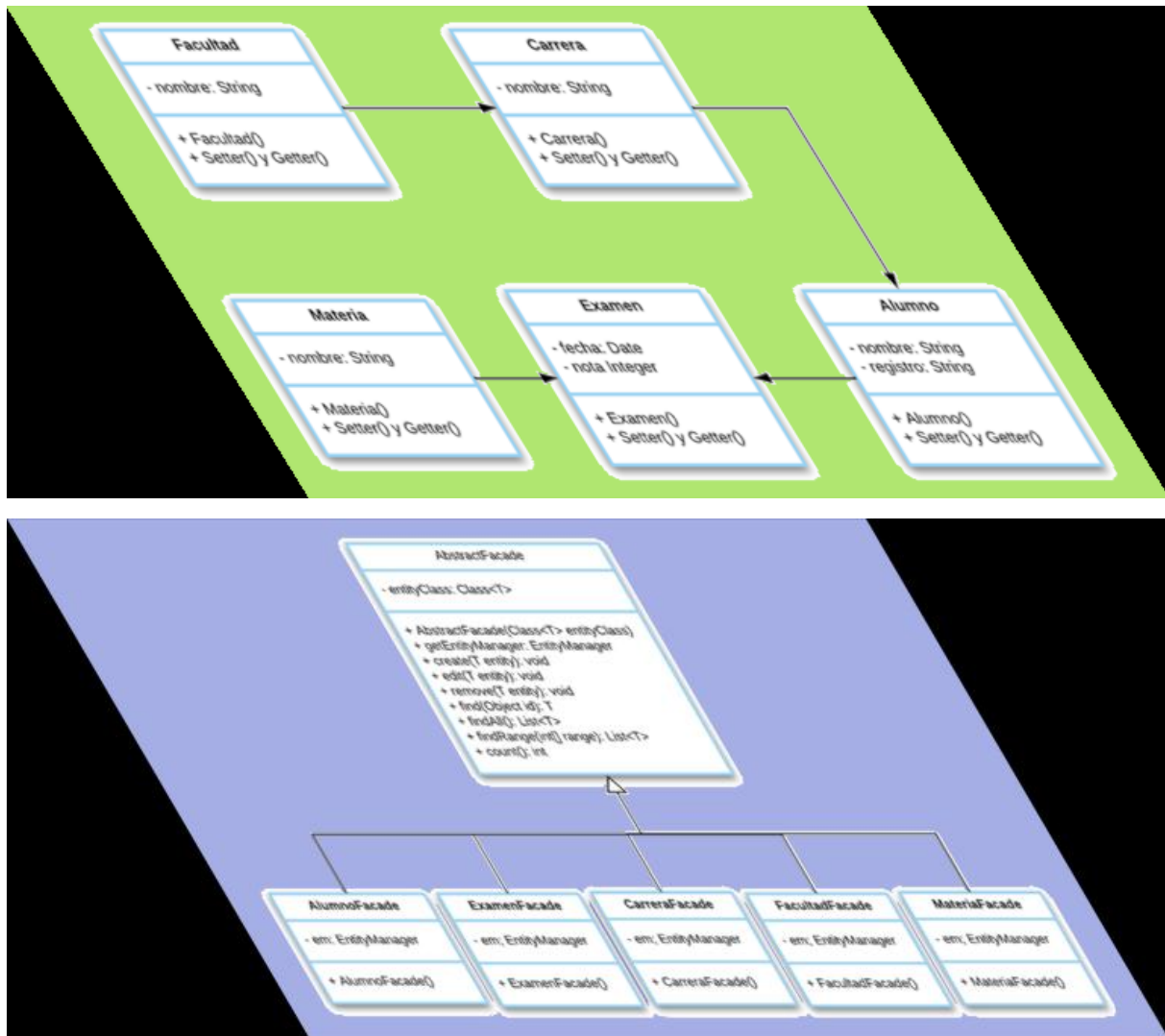
@Entity
public class Alumno {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nombre;
    private String apellido;

    // constructores, getters y setters
}
...
```

- Relaciones comunes:

- \* `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`
- \* Usar `@JoinColumn` para especificar la columna FK y `mappedBy` para relaciones bidireccionales.



## 2. Mapeo de Relaciones

### 2) Mapeo de relaciones entre entidades

- Uno a muchos (ejemplo Alumno -> Inscripcion):

```

@Entity public class Curso { @Id @GeneratedValue private Long id; private String nombre;
@OneToMany(mappedBy = "curso", cascade = CascadeType.ALL, orphanRemoval = true)
private List inscripciones = new ArrayList<>();
}

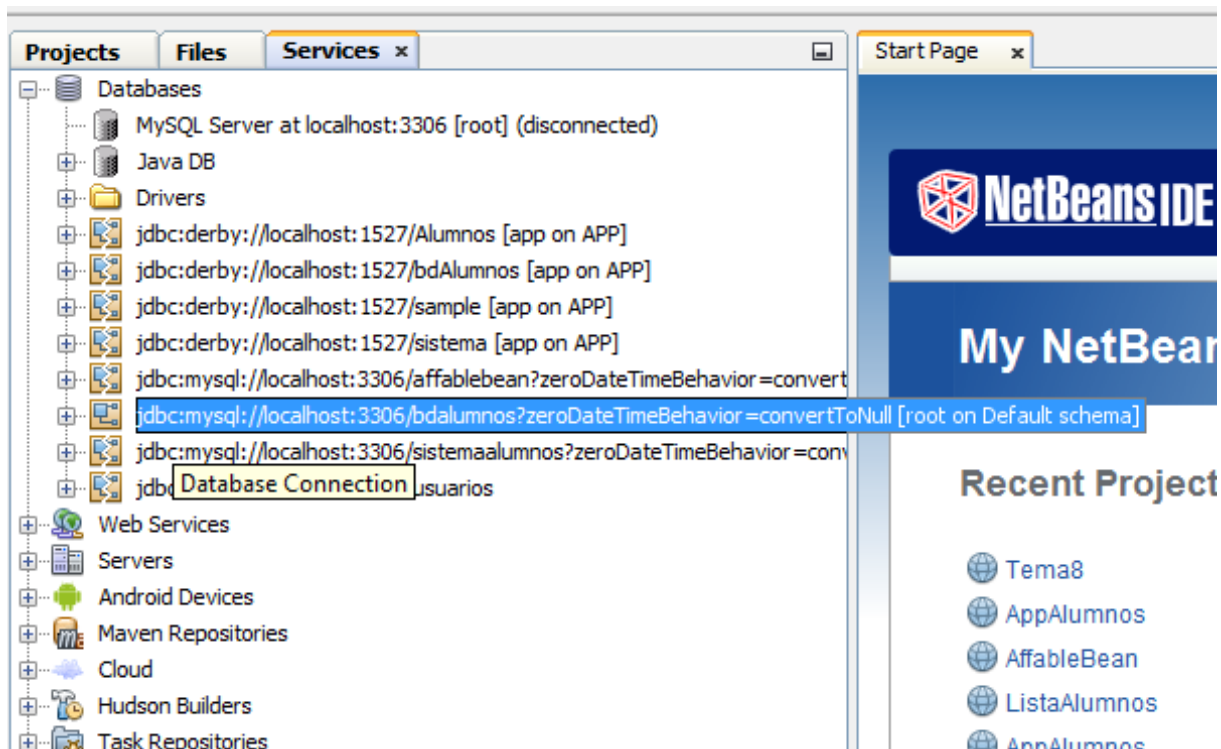
@Entity
public class Inscripcion {
    @Id @GeneratedValue
    private Long id;

    @ManyToOne
    @JoinColumn(name = "curso_id")
    private Curso curso;

    @ManyToOne
    @JoinColumn(name = "alumno_id")
    private Alumno alumno;
}

```

- Elegir FetchType (LAZY por defecto en colecciones) y entender CascadeType según comportamiento deseado.



### 3. Mapeo de Herencia

#### 3) Herencia en JPA

- Estrategias: SINGLE\_TABLE (por jerarquía), JOINED, TABLE\_PER\_CLASS.
- Ejemplo usando JOINED:

```
@Entity @Inheritance(strategy = InheritanceType.JOINED) public class Item { @Id
@GeneratedValue private Long id; private String titulo; }
```

```
@Entity
public class Libro extends Item {
private String isbn;
}
...
```



Home About...

User: admin Domain: domain1 Server: localhost

**Eclipse GlassFish**

Enable logging commands

General Advanced **Additional Properties**

**Edit JDBC Connection Pool Properties**

Modify properties of an existing JDBC connection pool.

Pool Name: BDAlumnosPool

**Additional Properties (8)**

☒ ☐ Add Property Delete 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

## 4. Crear Facades (EJB + JPA) - CRUD

4) Crear un Session Facade (Stateless EJB) para exponer operaciones CRUD

- El patrón Facade centraliza operaciones sobre entidades y reduce acoplamiento.
- Usamos @Stateless para un EJB sin estado y @PersistenceContext para inyectar el EntityManager.

Ejemplo de Facade genérico:

```
import javax.ejb.Stateless; import javax.persistence.EntityManager; import
javax.persistence.PersistenceContext; import java.util.List;

@Stateless
public class AlumnoFacade {

    @PersistenceContext(unitName = "AppBeansPU")
    private EntityManager em;

    public void create(Alumno a) {
        em.persist(a);
    }

    public Alumno find(Long id) {
        return em.find(Alumno.class, id);
    }
}
```

```

public Alumno update(Alumno a) {
return em.merge(a);
}

public void delete(Long id) {
Alumno a = em.find(Alumno.class, id);
if (a != null) em.remove(a);
}

public List findAll() {
return em.createQuery("SELECT a FROM Alumno a", Alumno.class).getResultList();
}
}
...

```

- Repetir para cada entidad: podés generar un Facade genérico (AbstractFacade) y luego extenderlo.

General
Advanced
Additional Properties

Ping Succeeded

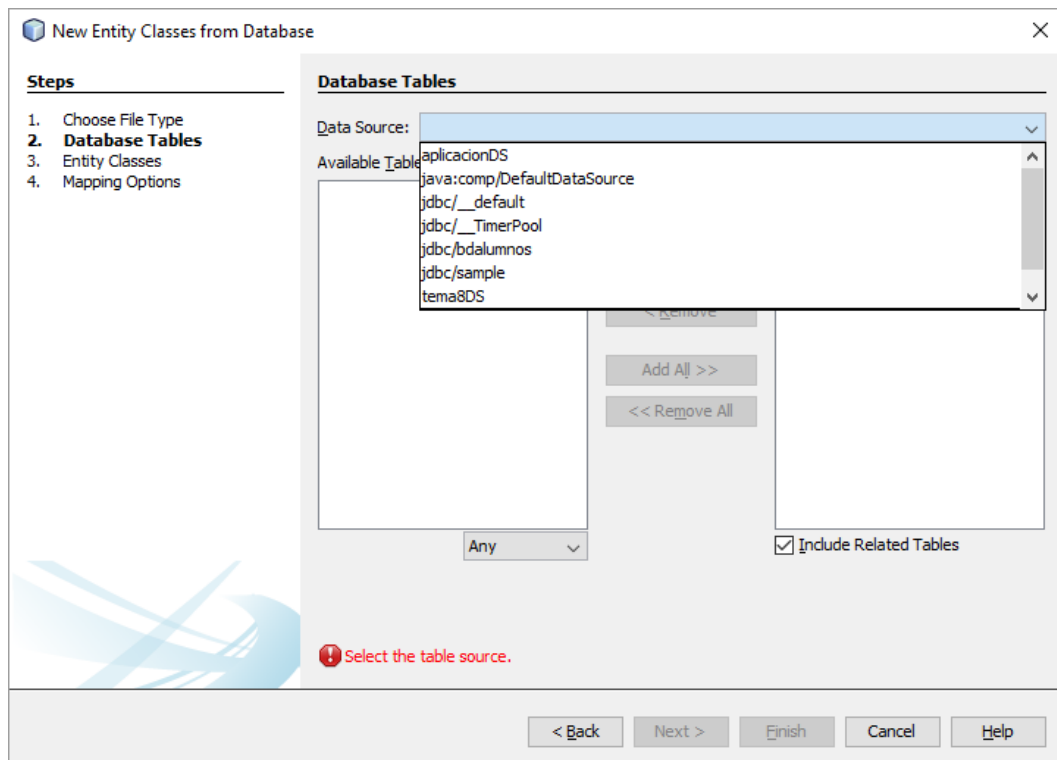
### Edit JDBC Connection Pool

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:	<div> javax.sql.ConnectionPoolDataSource </div> Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:	<div>com.mysql.cj.jdbc.MysqlDataSource</div> Vendor-specific classname that implements the DataSource and/or XADataSource APIs
Driver Classname:	<div></div> Vendor-specific classname that implements the java.sql.Driver interface.
Ping:	<input type="checkbox"/> When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attrib
Deployment Order:	<div>100</div> Specifies the loading order of the resource at server startup. Lower numbers are loaded first.
Description:	<div></div>



## 5. AbstractFacade: reutilizar código

### 5) AbstractFacade para evitar repetir código

```
import java.util.List; import javax.persistence.EntityManager;

public abstract class AbstractFacade {
    private Class entityClass;

    public AbstractFacade(Class entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

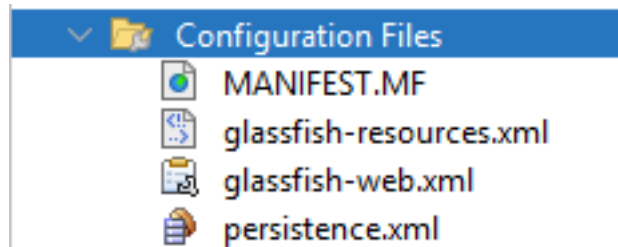
    public void create(T entity) { getEntityManager().persist(entity); }
    public void edit(T entity) { getEntityManager().merge(entity); }
    public void remove(T entity) { getEntityManager().remove(getEntityManager().merge(entity)); }
    public T find(Object id) { return getEntityManager().find(entityClass, id); }
    public List findAll() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }
    ...
}
```

Ejemplo de Facade concreto:

```
@Stateless public class AlumnoFacade extends AbstractFacade<Alumno> {
    @PersistenceContext(unitName = "AppBeansPU") private EntityManager em;

    protected EntityManager getEntityManager() { return em; }

    public AlumnoFacade() { super(Alumno.class); }
    ...
}
```



## 6. Buenas prácticas y pruebas

### 6) Buenas prácticas y pruebas

- Mantener DTOs si no querés exponer entidades directamente a la capa web.
- Manejar transacciones con EJB (por defecto EJB usa container-managed transactions).
- Añadir validaciones (@NotNull, @Size, Bean Validation) en las entidades.
- Escribir pruebas integradas que levanten el contenedor o usar pruebas con embeddable containers.