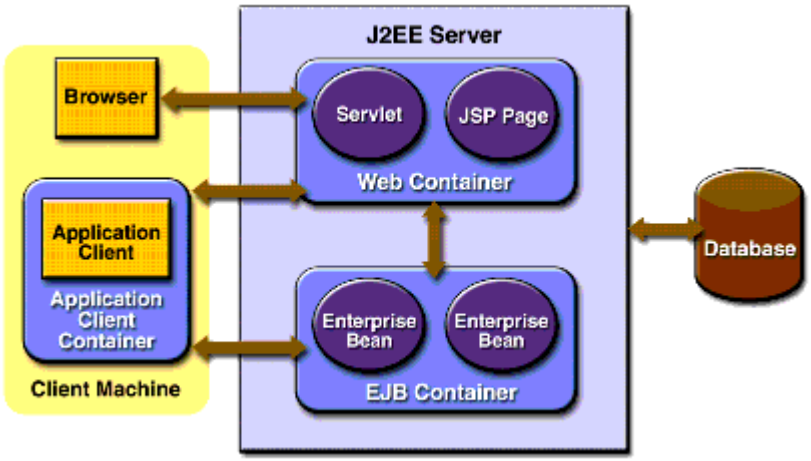
- Tipos de contenedores:
  - Contenedor de aplicación cliente
  - Contenedor Web
  - Contenedor de aplicación servidor



Sin servidor de aplicaciones

LOS VERDADEROS PROGRAMADORES PROGRAMAN EN BINARIO

Resolver problemas a nivel de:

-Transacciones
-Persistencia
-Distribución de objetos
-Construcción frameworks propios

Con servidor de aplicaciones

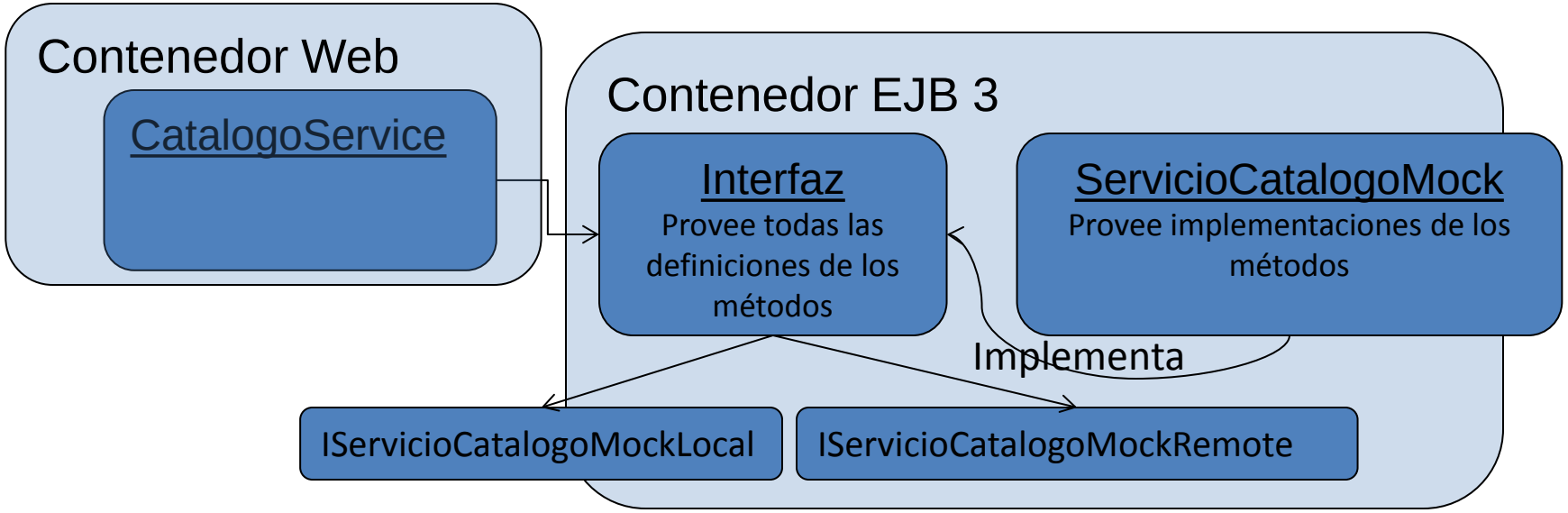
Provee servicios comunes como:

- Persistencia
- Transacciones
-Distribución de objetos
-Los proveedores implementan los estándares
-Permiten centrarse en los problemas del negocio

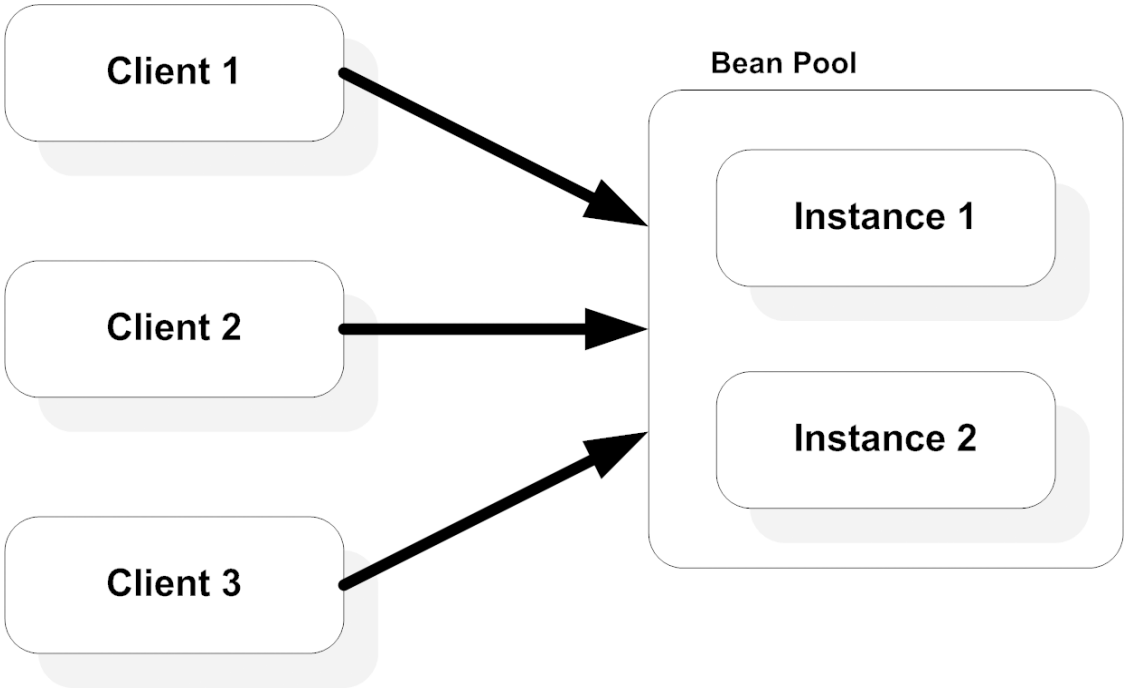
Session Bean

- Que son los componentes session bean?

  “Son una tecnología EJB que permiten encapsular procesos de negocio” (EJB 3 developer guide)

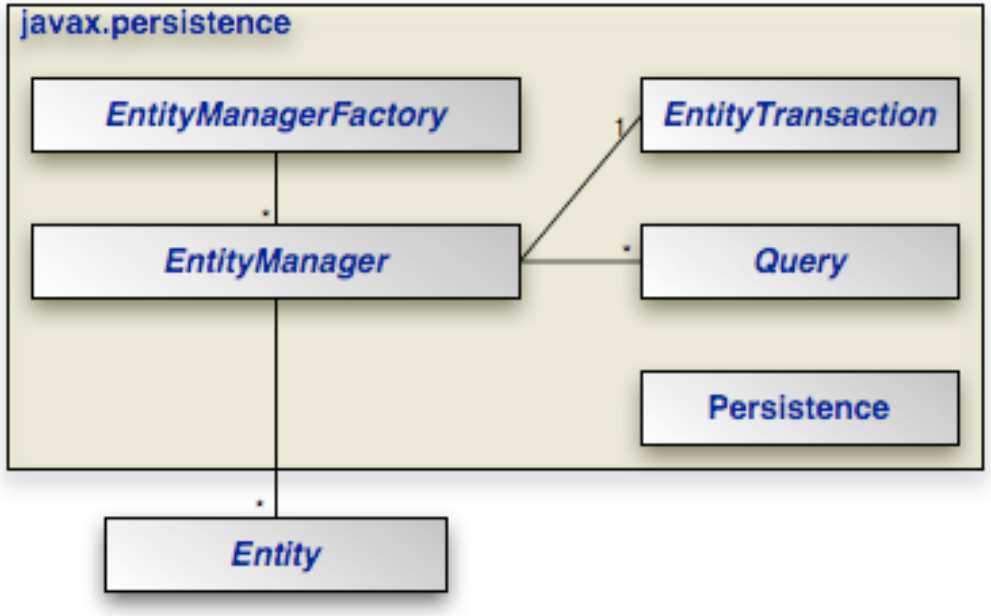


Stateless Session Bean



Entity beans

- Enterprise Java Beans 3.0 Persistence
  - Especificación creada por SUN
  - Persistencia de POJOs cualquier RDBMS
  - Uso de anotaciones y generics

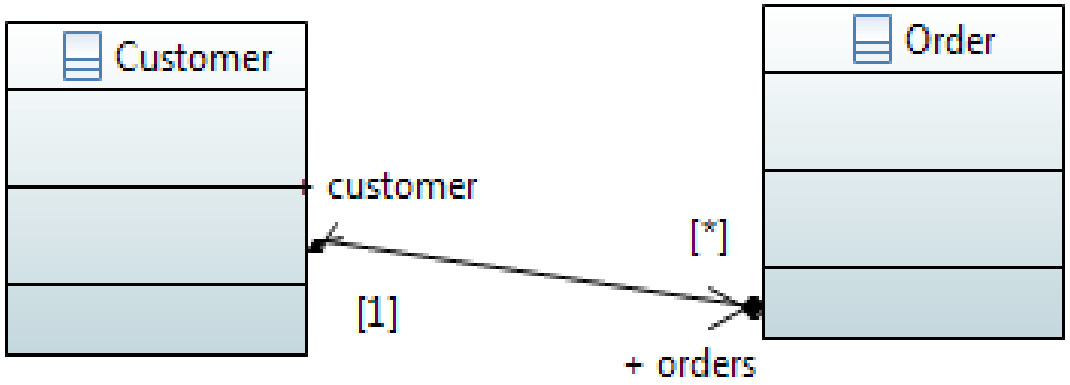


```
@Entity
public class Customer implements Serializable {
private Long id;
private String name;
private Address address;
// No argument constructor
public Customer() {}
@Id
public Long getID() {
return id;
}
private void setID (Long id) {
this.id = id;
} …
```

Relaciones entre entity beans

- Existen 4 tipos de relaciones
  - Uno a Uno
  - Uno a Muchos
  - Muchos a Uno
  - Muchos a Muchos

Ejemplo de relación Uno a Muchos



Customer.java

```
@OneToMany
public Collection<Order> getOrders()
```

Order.java

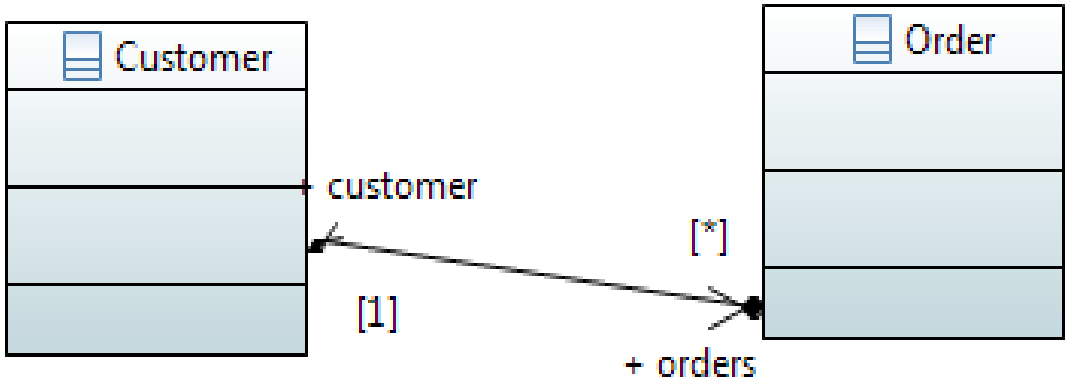
```
private Customer customer;

@ManyToOne
public Customer getCustomer()
```

Relaciones bidireccionales

- Se puede navegar en ambas direcciones
- Deben cumplir las siguientes reglas
  - El lado inverso de la relación debe referenciar el lado dueño usando el elemento *mappedBy* en la anotación de la relación

Ejemplo de relación bidireccional



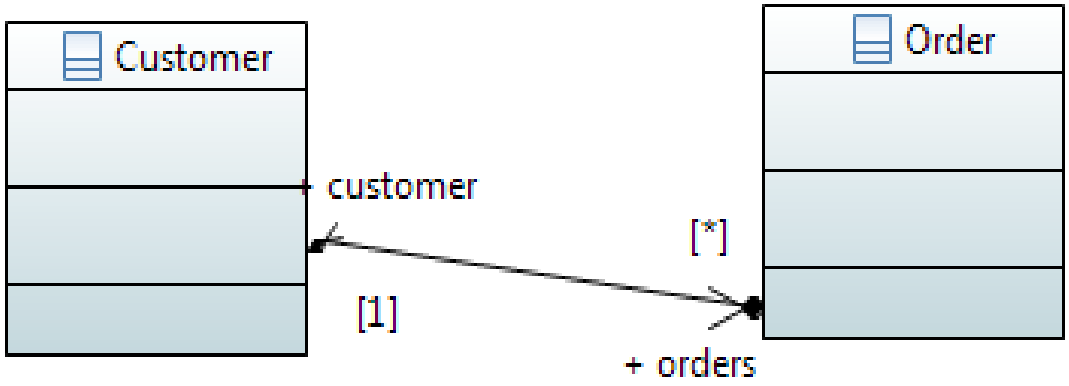
La aplicación requiere ambas consultas:

- Obtener las ordenes para un cliente
- Obtener el cliente para una orden

¿Cómo usar el mappedBy?

- Se necesita identificar quién es lado dueño y el lado inverso
- El mappedBy se coloca en lado inverso
- En la relación “muchos a uno” el lado dueño es el lado de muchos
- En las relaciones “uno a uno” o “muchos a muchos” el dueño depende de a dónde quiero colocar la llave foránea

¿Quién es el dueño de la siguiente relación bidireccional en OO?

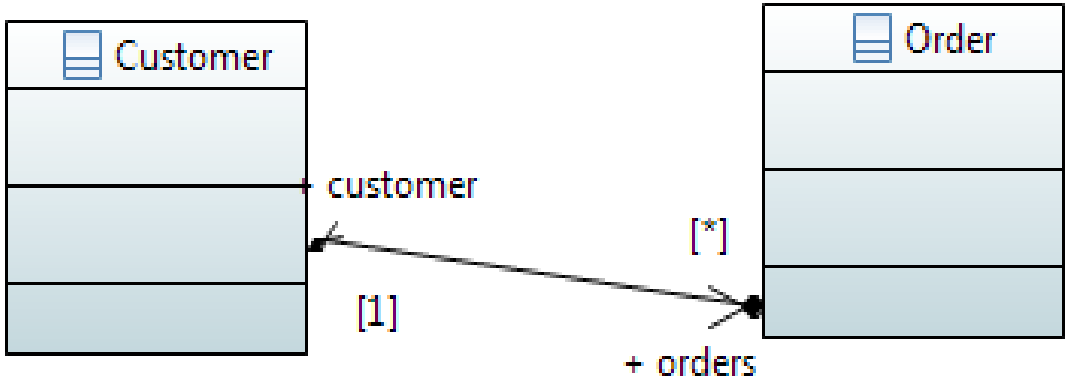


R:/
Parecería
el
Customer

Lo opuesto al mundo OO

- En el mundo OO
  - El customer posee orders (orders son una lista de customer)
  - No hay una orden sin customer
  - El customer parece ser el dueño de las órdenes
- En el mundo SQL
  - Hay registros que tienen apuntadores a otros
  - Dado que para un customer hay N orders, cada orden contiene a FK
  - Esta FK es una conexión que significa que la orden posee (o literalmente contiene) la conexión
  - El lado dueño es order

¿Quién es el dueño de la siguiente relación bidireccional en SQL?



Customer

Order
Customer_FK

R:/ Orden

Dónde se coloca el mappedBy

- El mappedBy se coloca en lado inverso, i.e, Customer

Customer.java

```
@OneToMany (mappedBy = "customer")
public Collection<Order> getOrders()
```

Order.java

```
private Customer customer;

@ManyToOne
public Customer getCustomer()
```

- En el ej., anotaciones están puestas en getters

Carga en memoria de las relaciones, i.e., fetch type

- Esto aplica tanto para relaciones unidireccionales como bidireccionales
- Indica el momento en el cual los datos asociados al campo relacionado con otra entidad son cargados en memoria
  - EAGER: cuando se carga la clase dueña que tiene la relación
  - LAZY: cuando se utiliza el campo que representa la relación
- *Fetch* type por defecto es *LAZY*

Ejemplo en bidireccional

```
public class Customer {
@OneToMany (fetch=FetchType.EAGER, mappedBy="order")
private Collection<Order> orders;
}
```

```
public class Order {
@ManyToOne(fetch = FetchType.LAZY)
protected Customer customer;
}
```

Consultas

- createQuery

em.createQuery("SELECT c FROM Customer c WHERE c.name LIKE :custName") .setParameter("custName", name) .setMaxResults (10).getResultList();

- createNamedQuery
  - Declaración
- **@NamedQueries(value={@NamedQuery (name="findAllCustomersWithName", query="SELECT c FROM Customer c WHERE c.name LIKE :custName")**

- Uso
  - **em.createNamedQuery("findAllCustomersWithName") .setParameter("custName", "Smith") .getResultList();**

Unidades de persistencia

- Conjunto de todas las clases administradas por el
- EntityManager instances en la aplicación
- Definidas en el archivo de configuración persistence.xml