Desarrollo de Sistemas de Información

Tema 2. Diseño Conceptual

Marta Elena Zorrilla Pantaleón

Dpto. de Matemáticas, Estadística y Computación

Diseño Conceptual

- En sistemas de información se asume que un dominio consiste de un número de objetos y las relaciones entre ellos que se clasifican en conceptos
  - Conceptos: cliente, producto y venta
    - Objetos: cliente y producto
    - Relación: venta
- Un modelo conceptual incluye no sólo objetos, relaciones y conceptos (**estructura)** sino también cómo el dominio cambia (**comportamiento**). Además debe ser consistente (**reglas de integridad**) y puede ser necesario recoger **reglas de derivación**.

Diseño conceptual: estructural

- Un **concepto** es algo que se ha formado en nuestra mente a través de la generalización de ciertas instancias.
  - La extensión de un concepto es el conjunto de instancias posible
  - La intensión de un concepto es el conjunto de propiedades compartidas de las instancias.
- **Tipo de entidad** es un concepto cuyas instancias son objetos identificables. Toda instancia es de un tipo de entidad, aunque pudiera serlo de más de una (persona, estudiante)
- **Tipos de relación** son conceptos cuyas instancias son relaciones entre dos o más entidades (instancias de un tipo de entidad)
- Todo aquello en lo que estemos interesados debe estar conceptualizado

Diseño Conceptual: comportamiento

- Eventos de dominio (cambios en dominio)
  - Cuando se produce un cambio de una o más instancias de un tipo entidad o relación
  - Ej. Transferencia bancaria
- Acciones
  - Consultas (no producen cambio en el dominio)
  - Acción temporal que ocurre independiente del estado del sistema
  - Acción explícita iniciada por una condición satisfecha (restricciones de negocio, modos de operación del negocio, etc.)
- Eventos y acciones son instancias de conceptos. Tiene características que son las relaciones con otras entidades.
  - P.ej. Las características de una transferencia son la cuenta origen y destino y la cantidad transferida

Diseño Conceptual: restricciones y reglas

- Restricciones de integridad
  - Requerido
  - Único
  - Posible conjunto de valores
  - Cardinalidad
  - Restricciones de dominio no estructurales
- Reglas de derivación
  - Cómo inferir hechos nuevos partiendo de otros

Ingeniería de Requisitos

- El modelado conceptual precede al diseño del sistema
- Un documento con los requisitos del sistemas precede al modelado conceptual.
- Ingeniería de requisitos es un proceso complejo porque implica la participación de diferentes actores (usuarios, analistas, jefes, …) con diferentes puntos de vista, necesidades e intereses.
  - Definir las características del dominio y las funciones que debe realizar el sistema de información (1er nivel de detalle)
  - Definir los requisitos funcionales y no funcionales (denominada especificación de requisitos)
  - Apoyarse en modelos conceptuales para validar con los usuarios los requisitos antes de pasar al diseño (esquemas conceptuales sencillos resultan útiles en esta fase)

ER para diseño de BD

EVOLUCIÓN HISTÓRICA

- Propuesto por **Chen** en dos artículos ya históricos, en 1976 y 1977.
- Según Chen, *"El Modelo E/R puede ser usado como una base para una vista unificada de los datos"*, adoptando "***el enfoque más natural del mundo real que consiste en entidades y relaciones***".
- Posteriormente otros autores lo han ampliado con importantes aportaciones, formándose en realidad una familia de Modelos de Datos.
- Se abordará el modelo **E/R básico** y el modelo **E/R extendido**.
- El Modelo E/R ha tenido una gran difusión en la comunidad informática dedicada a las BD, prueba de ello es que ha sido el modelo más extendido en las herramientas CASE de ayuda al diseño de BD.
- DATE critica al modelo ER diciendo que no es más que una fina capa por encima del modelo relacional

Elementos estáticos

- Entidad (entity)
  - Objeto que existe y se distingue de los demás.
  - Pueden ser concretos
    - P. ej.: un libro, una persona,..
  - O abstractas
    - P.ej.: préstamo, pedido,…
- Atributo (attribute)
  - Propiedades que caracterizan a las entidades.
  - Clave primaria: atributos que identifican a la entidad
    - P.ej.: ISBN (PK), título, idioma,… para entidad libro
- Dominio (domain)
  - Conjunto de valores permitidos para un atributo
    - P. ej: indicando el tipo de datos (por intensión)
    - P. ej: sexo-> M o F (por extensión)

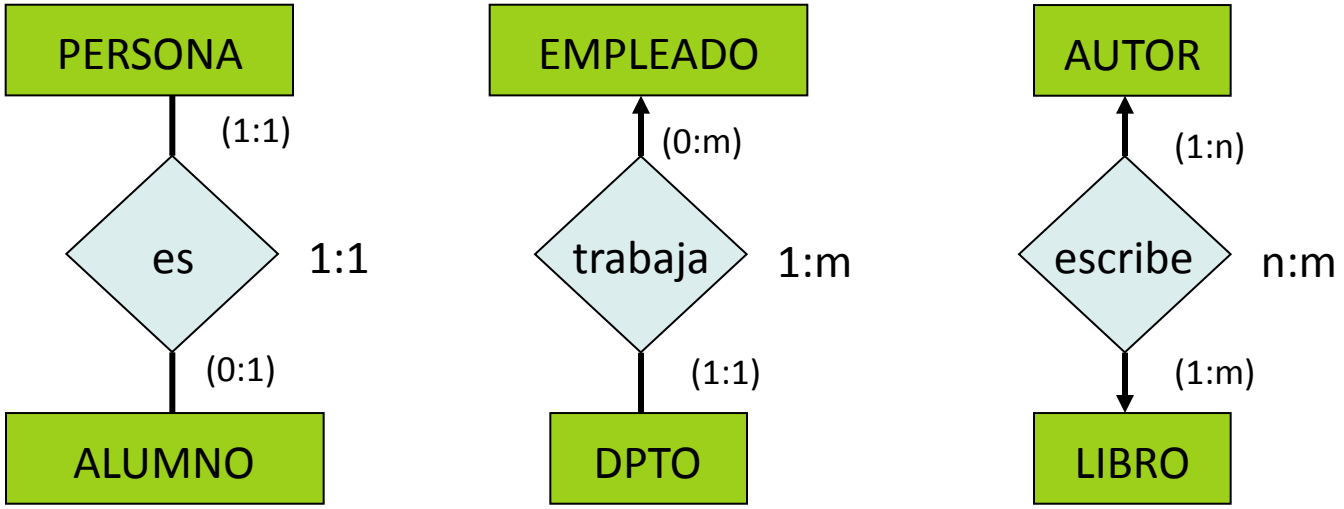
ENTIDADES

- Existen dos categorías de tipos de entidades:
  - Regulares o fuertes, que son aquellas cuyos ejemplares tienen existencia por sí mismos
    - Caso préstamos de la biblioteca: LIBRO y AUTOR

  - Débiles, en las cuales la existencia de un ejemplar depende de que exista un cierto ejemplar de otro tipo de entidad
    - Caso del EJEMPLAR que depende de LIBRO

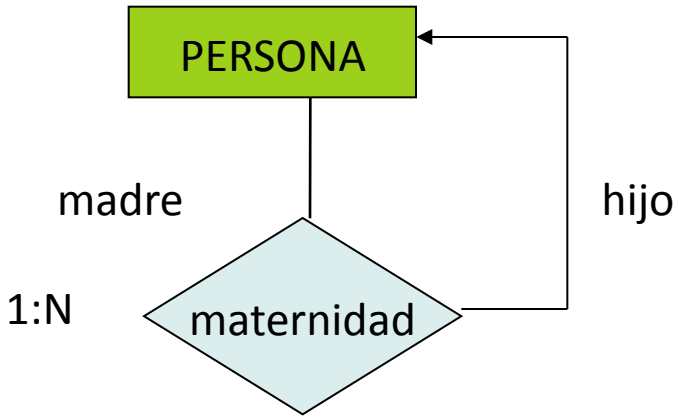
Elementos estáticos (y 2)

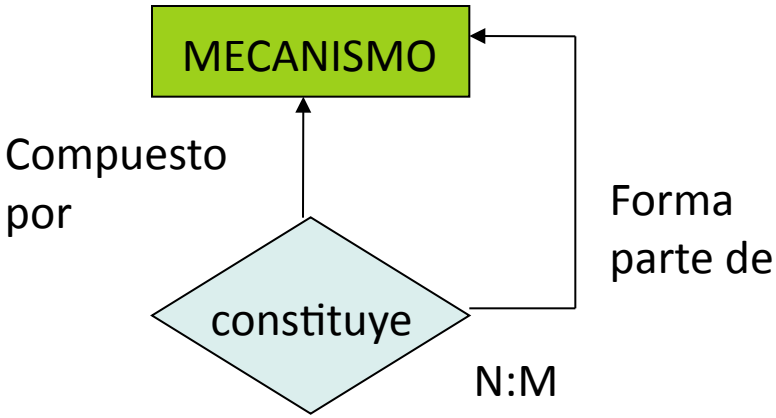
- Relación (relationship)
  - Conexión semántica entre dos o más entidades
  - Cardinalidad: nº máximo de unidades de un conjunto que se conecta o relaciona con una entidad de otro y viceversa



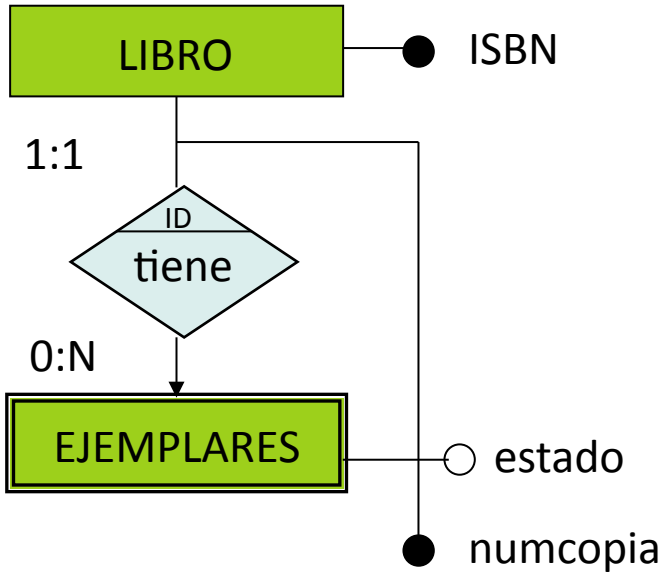
Relaciones reflexivas

- En estos casos se requiere especificar el rol, papel que desempeña en la relación

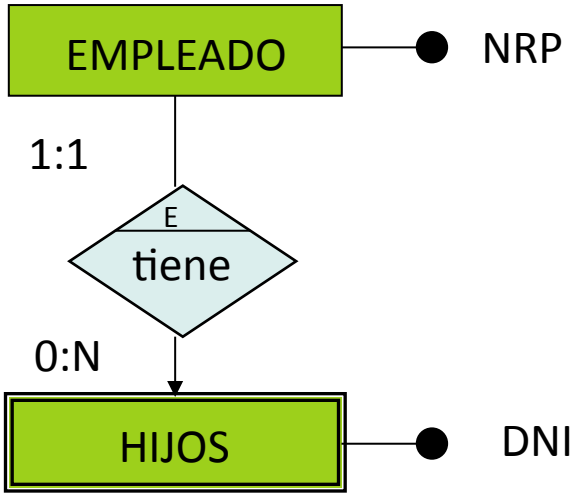




RELACIONES

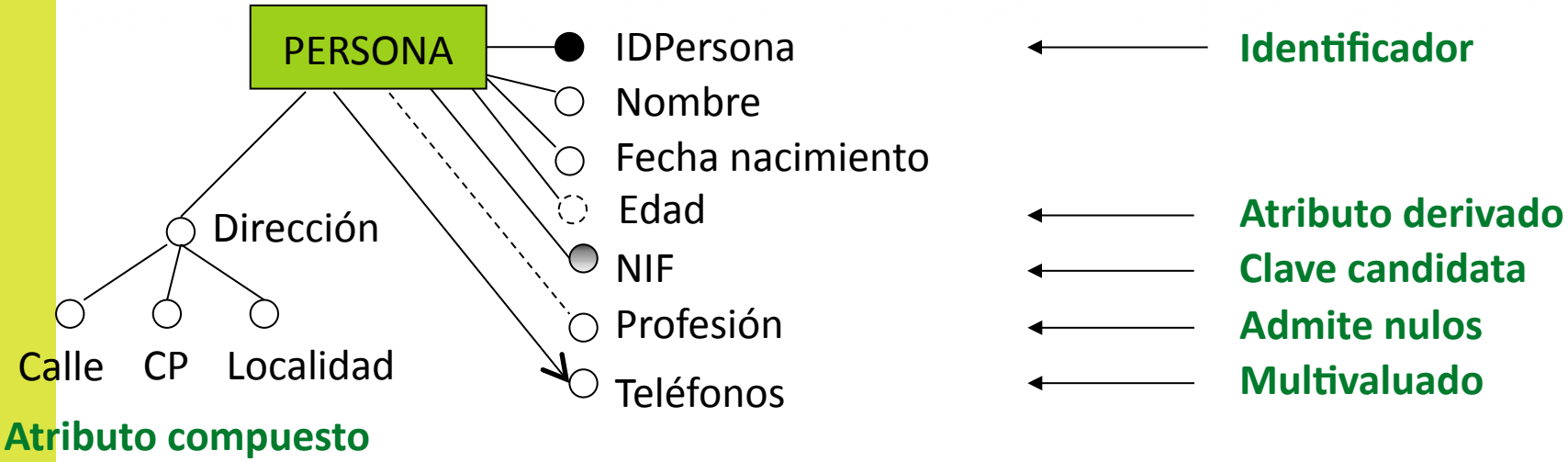


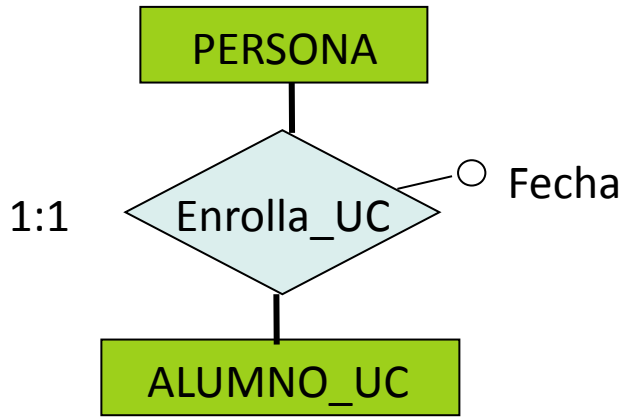
Dependencia de identificación

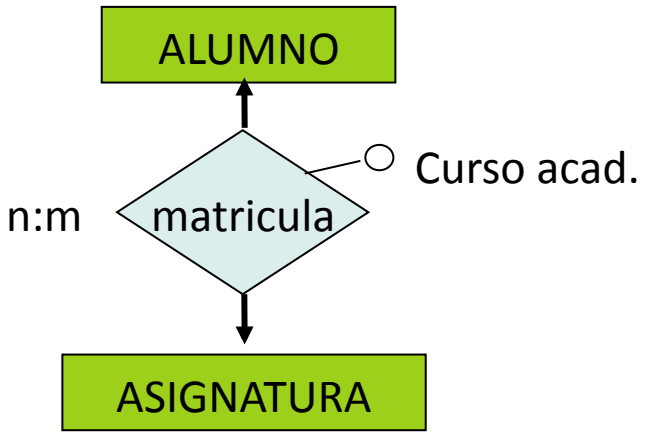


Dependencia de existencia

ATRIBUTOS Y CLAVES



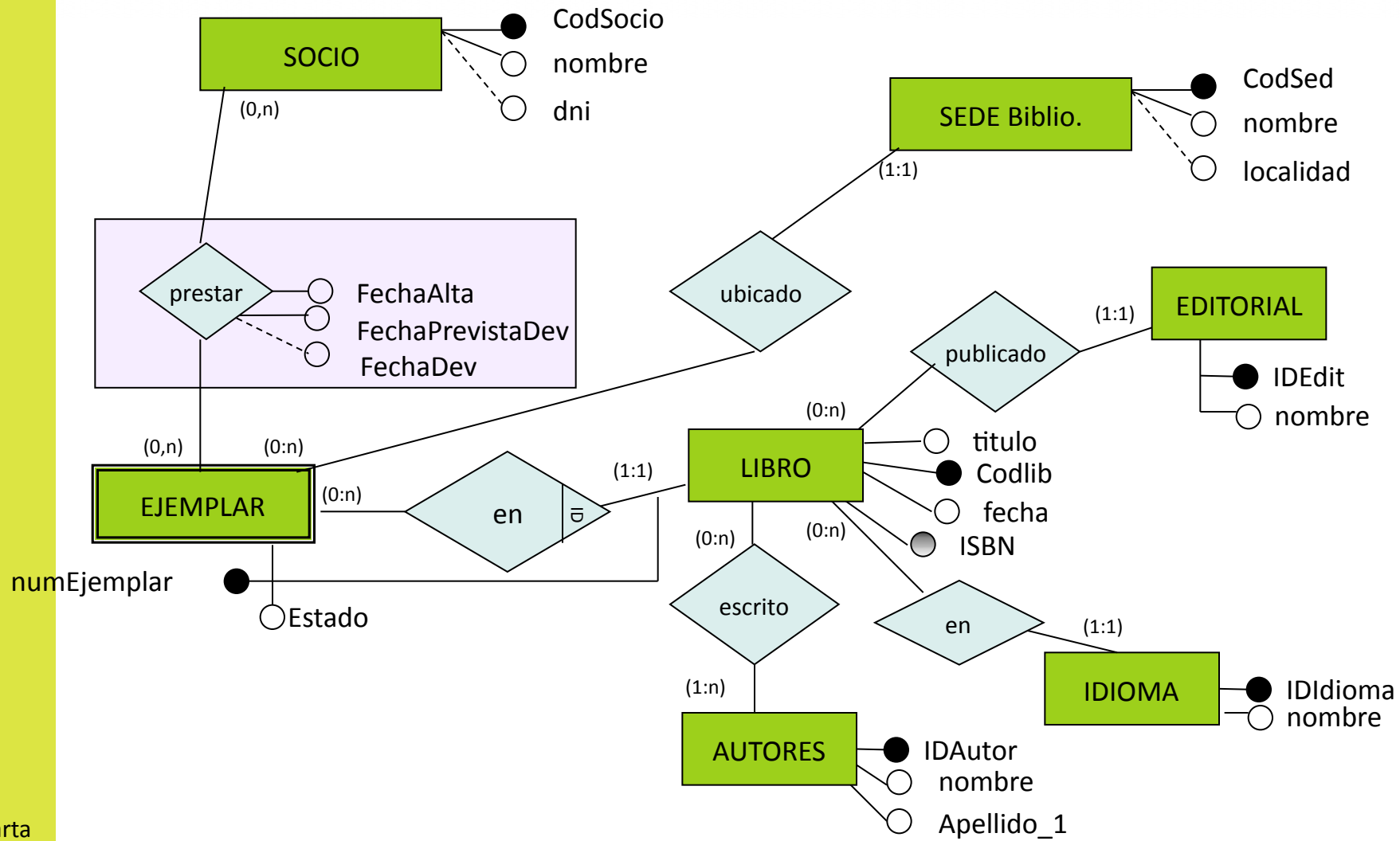




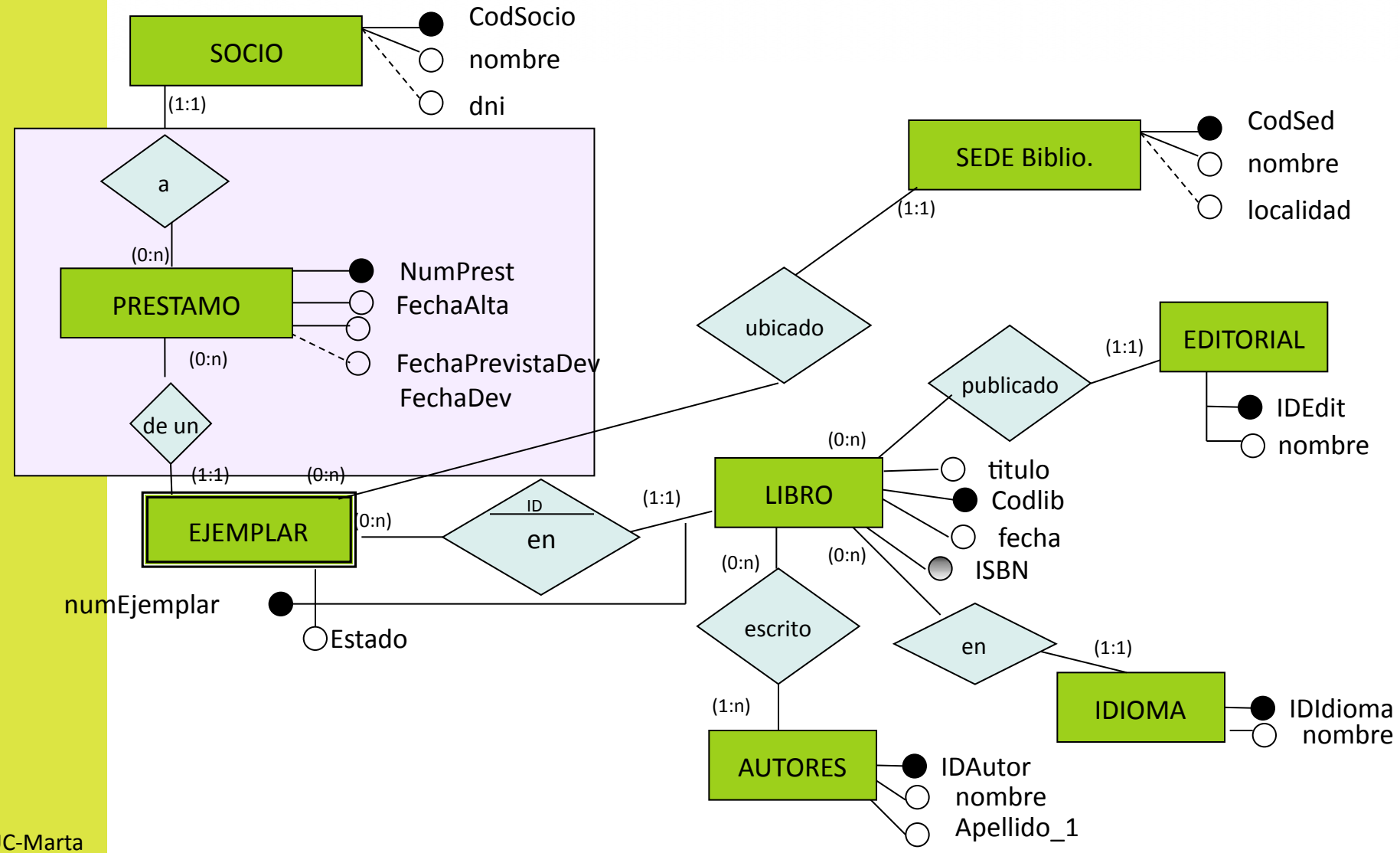
Gestión de préstamos (ejemplo)

- Requisitos:
  - La biblioteca está interesada en automatizar la gestión de préstamos
  - El modo de funcionar es sencillo, básicamente requiere registrar el socio que se lleva el libro, y de qué ejemplar se trata, así como las fechas de entrega, devolución prevista y de devolución.
  - La biblioteca está organizada en diversas sedes y el socio puede coger libros de cualquiera de ellas.
  - Del socio se tienen los datos personales básicos.
  - Y de los libros, todos los campos descriptivos que los caracterizan (título, idioma, autores, editorial, fecha,…).
  - Además de cada ejemplar se querrá conocer el estado en el que se encuentra (prestable, en reparación, fuera de circulación)

PRÉSTAMOS DE LA BIBLIOTECA



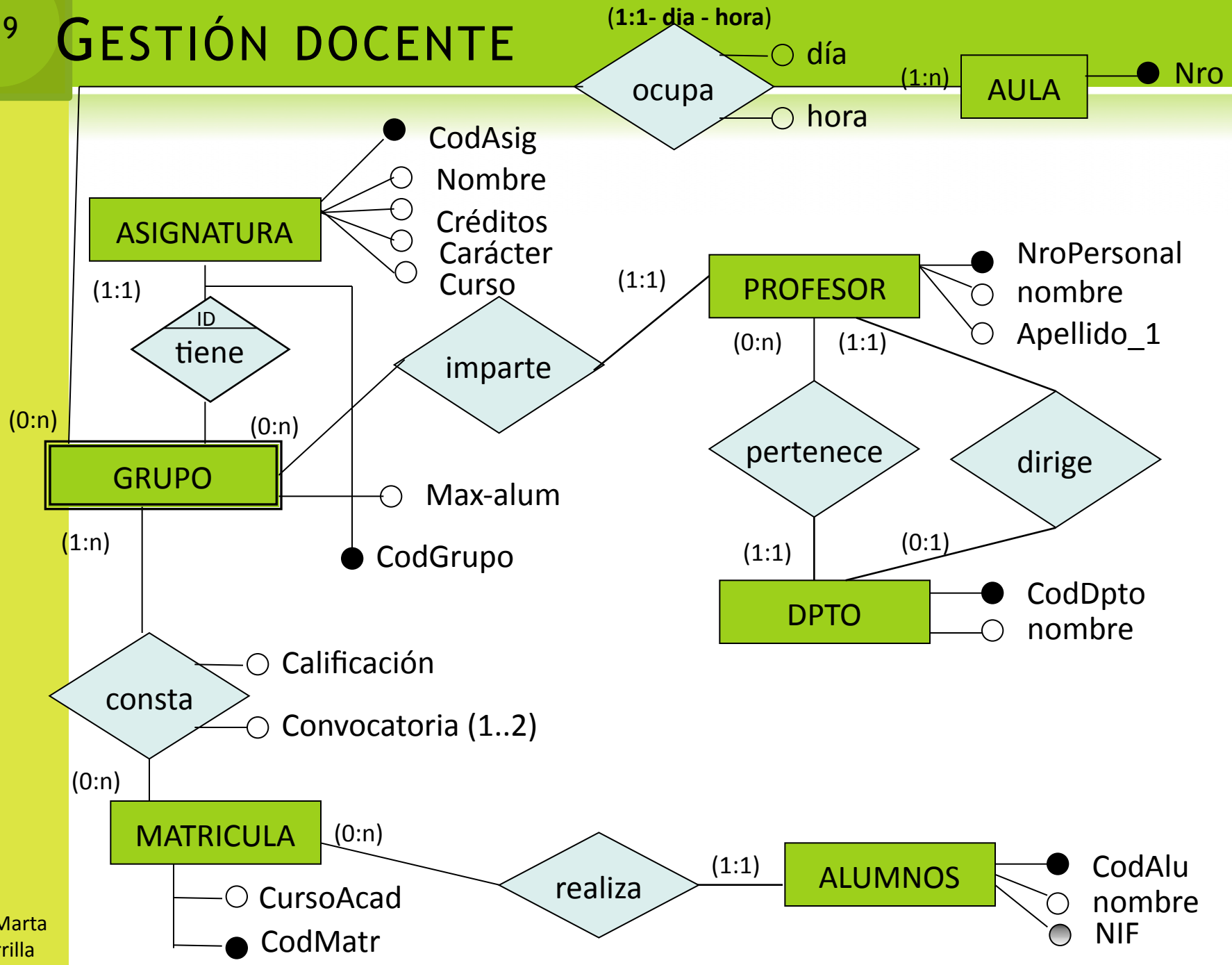
PRÉSTAMOS DE LA BIBLIOTECA



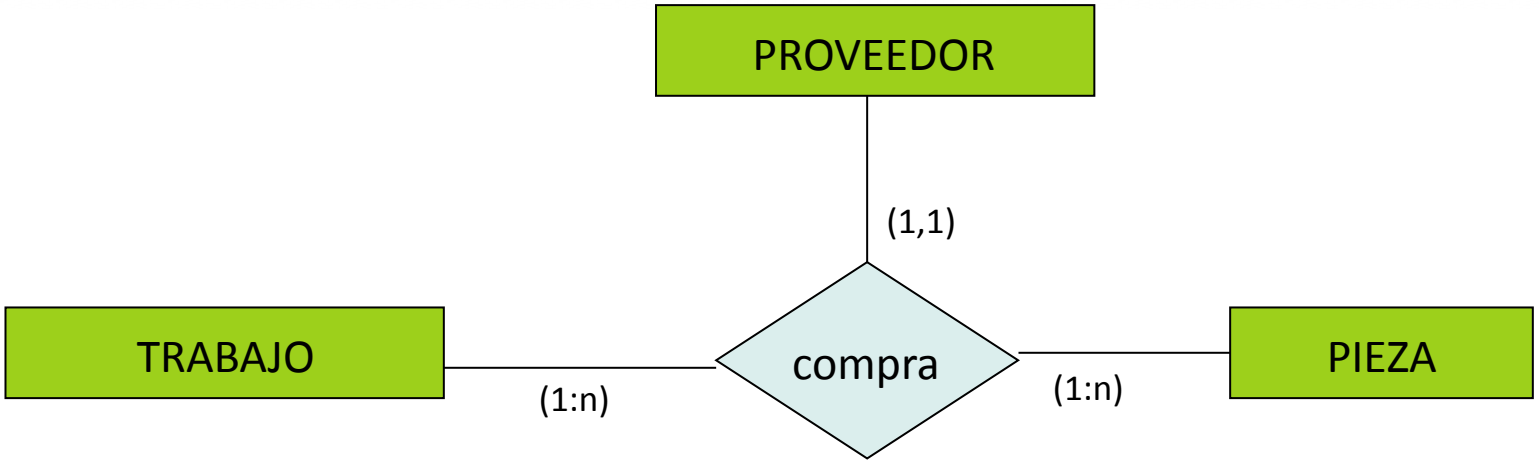
Gestión docente (ejemplo)

- Requisitos:
  - Cada profesor pertenece a un sólo departamento y debe pertenecer a uno.
  - El profesor puede impartir varios grupos de la misma o distinta asignatura, y un grupo debe ser enseñado por un profesor.
  - Los alumnos se matriculan de varias asignaturas (al menos una) cada curso académico pero han de hacerlo en un grupo (turno de mañana/tarde). A su vez un grupo tendrá varios alumnos matriculados. Cada grupo tendrá asignado un aula para cada día y hora de la semana.
  - La matrícula dará opción a dos convocatorias de examen con su respectiva calificación.
  - Todo departamento debe tener un director, que es profesor.
  - Los atributos de cada entidad son los que cabría esperar.

GESTIÓN DOCENTE

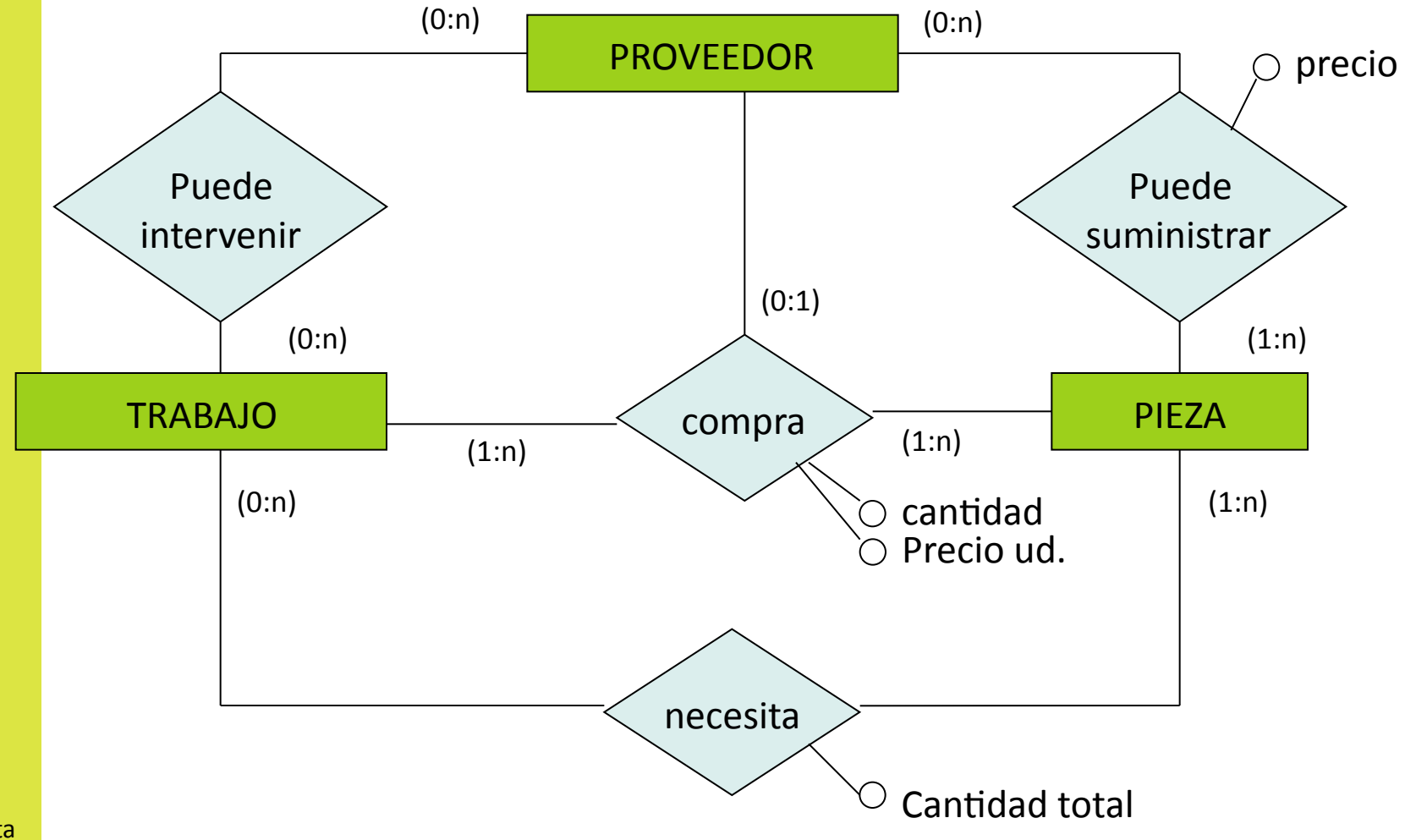


RELACIONES N-ARIAS

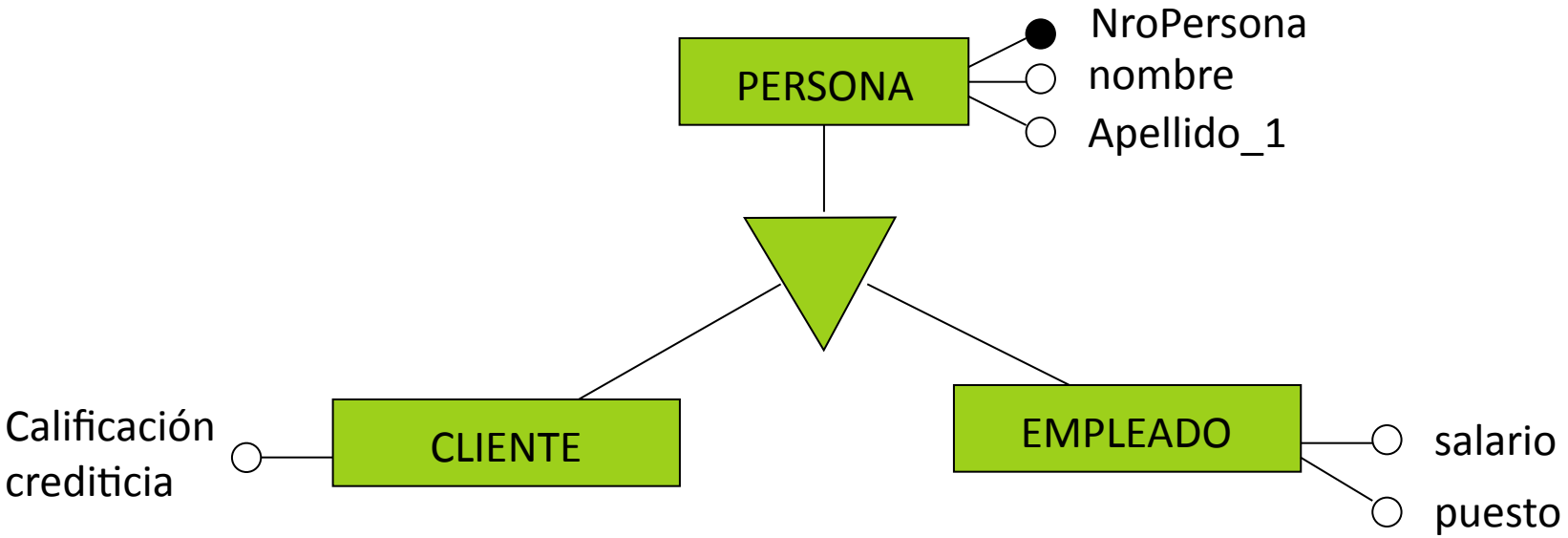


- Una pieza Y en un trabajo Z – una pareja (pieza, trabajo) – la suministran 0 o 1 proveedores.
- Un proveedor X en un trabajo Z – una pareja (proveedor, trabajo) – suministra 1, 2, .., n piezas.
- Un proveedor X suministra una pieza Y – una pareja (proveedor, pieza) – en 1, 2, .., n proyectos

RELACIONES N-ARIAS (SIN REDUNDANCIA)



Generalización y especialización



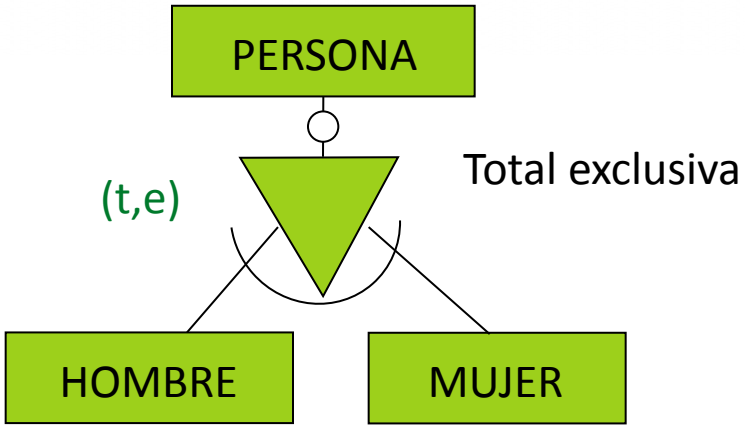
La **Generalización** se considera como un caso especial de relación entre uno o varios tipos de entidad (subtipos) y un tipo más general (supertipo), cuyas características son comunes a todos los subtipos.

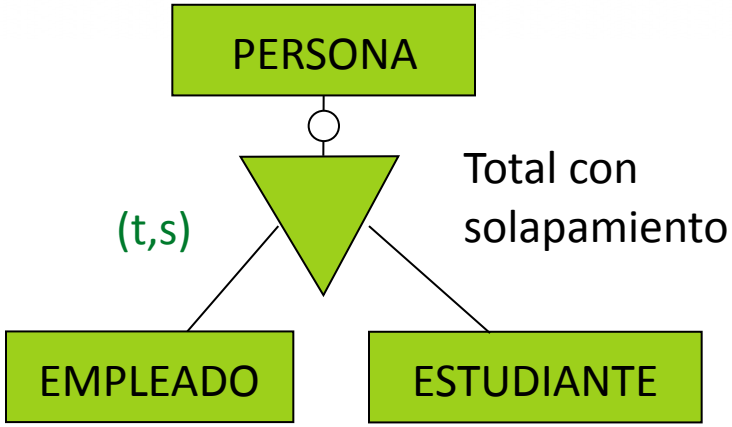
El mecanismo de abstracción contrario se llama **especialización**.

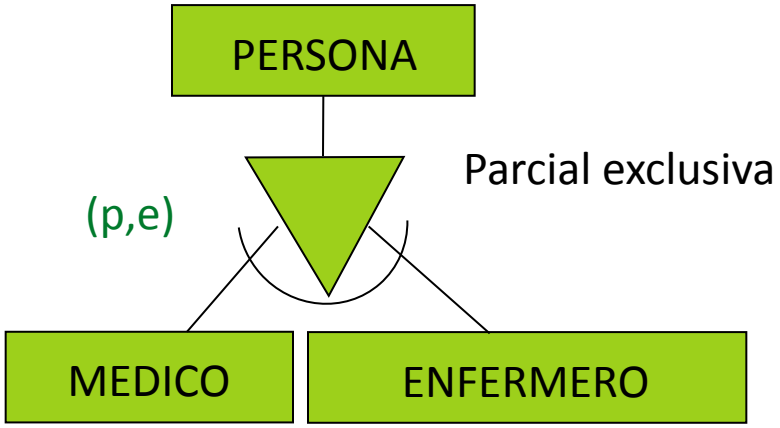
Generalización y especialización

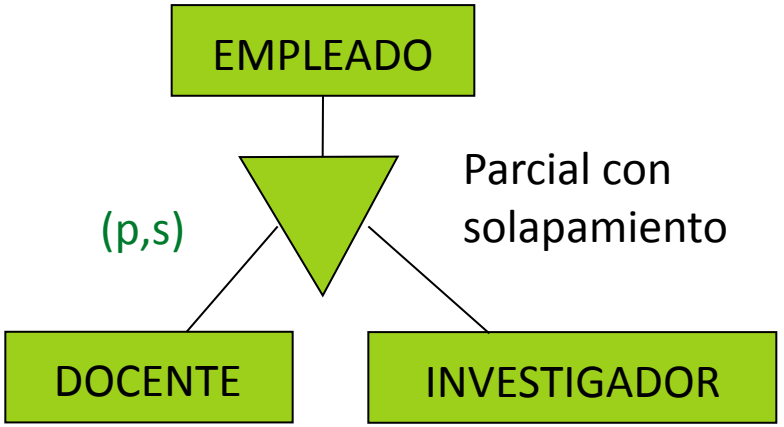
- La división en subtipos (especialización) puede venir determinada por una condición predefinida (por ejemplo, en función de los valores de un atributo llamado *discriminante*).
- La **Generalización/Especialización** tiene dos restricciones semánticas asociadas:
  - **Totalidad** (todo ejemplar del supertipo tiene que pertenecer a algún subtipo). El caso contrario se llama Parcialidad.
  - **Solapamiento** (un mismo ejemplar del supertipo puede pertenecer a más de un subtipo). El caso contrario se llama Exclusividad.

Generalización y especialización

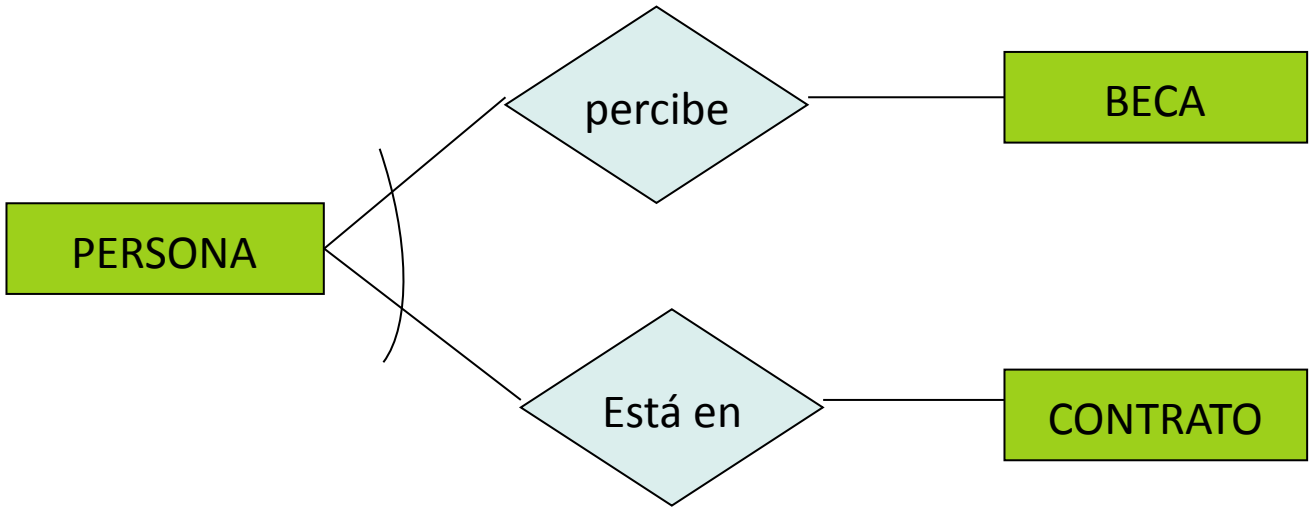






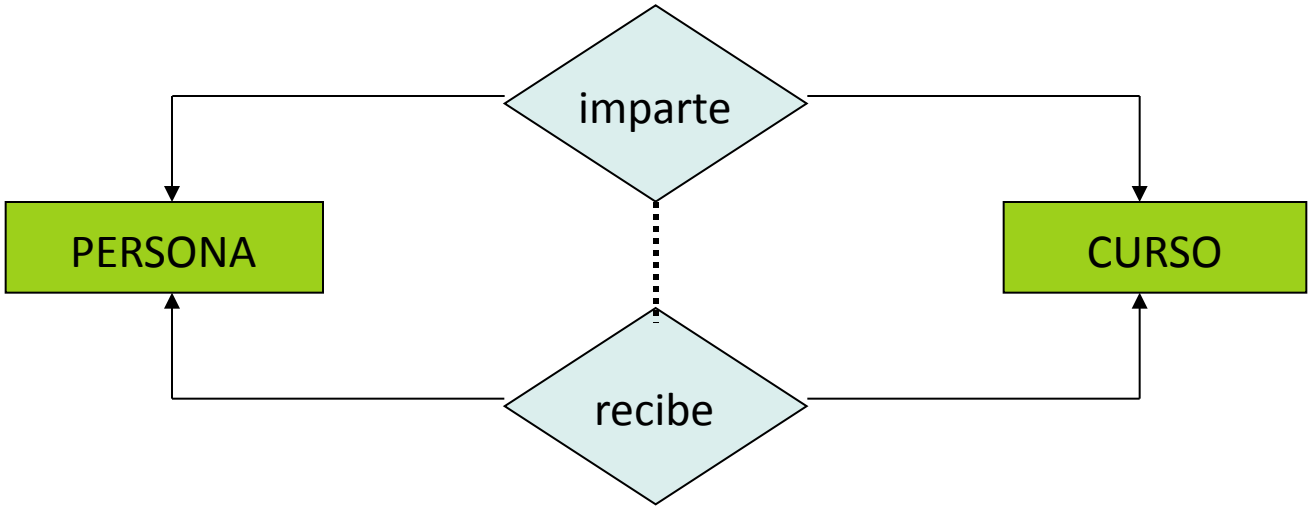


RESTRICCIÓN DE EXCLUSIVIDAD



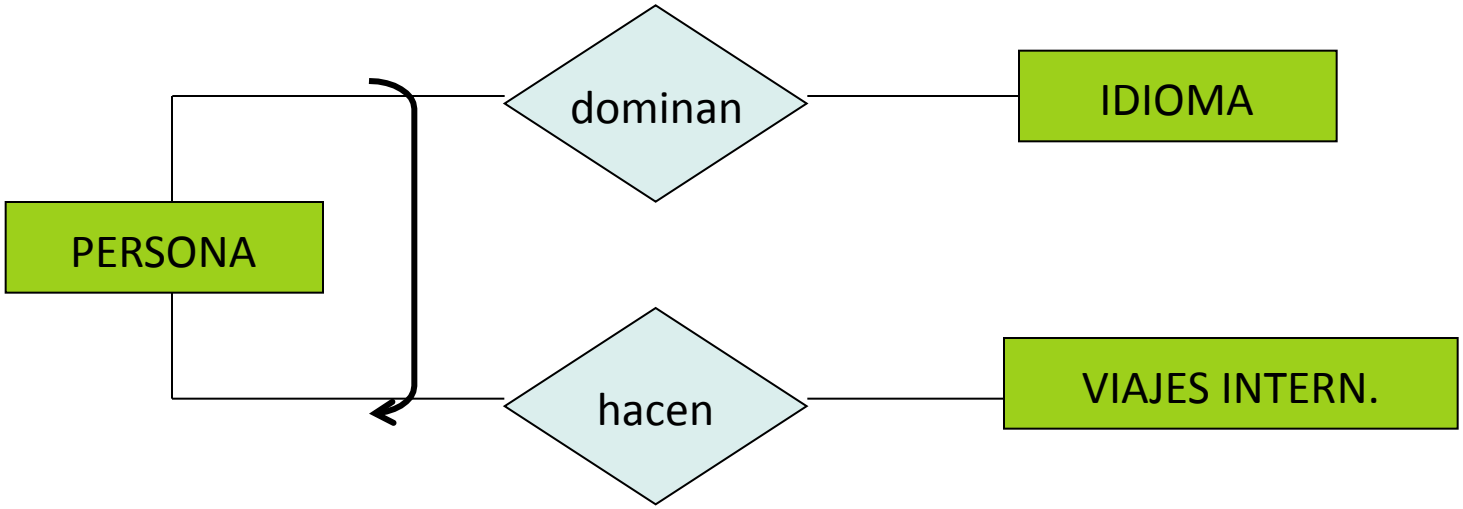
La persona o percibe una beca o está contratado

Restricción de exclusión



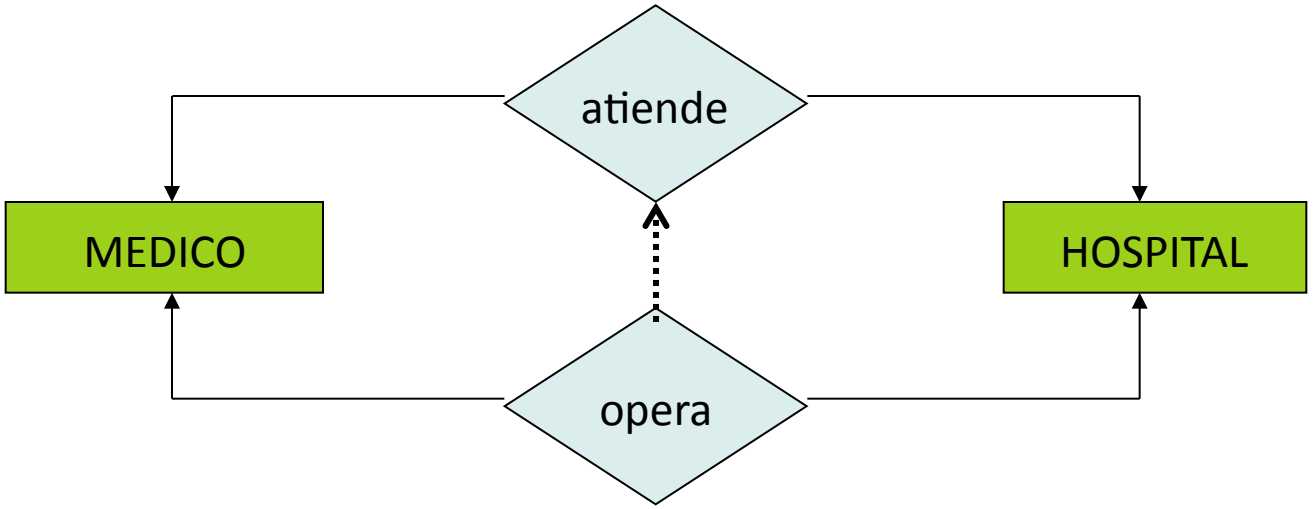
La persona imparte o recibe el curso, no puede estar en ambas relaciones a la vez

Restricción de inclusividad



Las personas que dominan idiomas son un subconjunto de las que realizan viajes internacionales. Si una persona participa en domina, tiene necesariamente que participar en hacen viajes

Restricción de inclusión



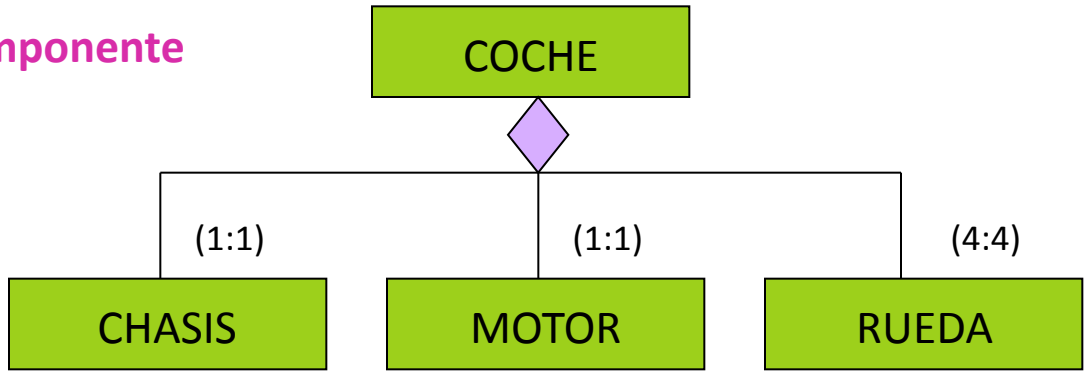
Los cirujanos son un subconjunto de los médicos del hospital

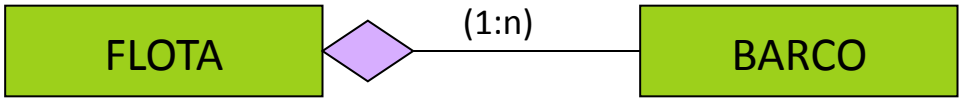
Agregación

- Es un tipo especial de relación en la cual las cardinalidades mínima y máxima del tipo de entidad agregada siempre son (1,1)
- Existen dos clases de agregaciones:
  - **Compuesto/Componente**:
    - permite representar que un "todo" se obtiene por la unión de diversas partes que pueden ser tipos de entidades distintas y que juegan diferentes roles en la agregación.
  - **Miembro/Colección**:
    - permite representar un "todo" como una colección de miembros, todos de un mismo tipo de entidad y todos jugando el mismo rol.
    - Esta agregación puede incluir una restricción de orden de los miembros dentro de la colección (indicando el atributo de ordenación).

Ejemplos agregación

Agregación
Compuesto/Componente



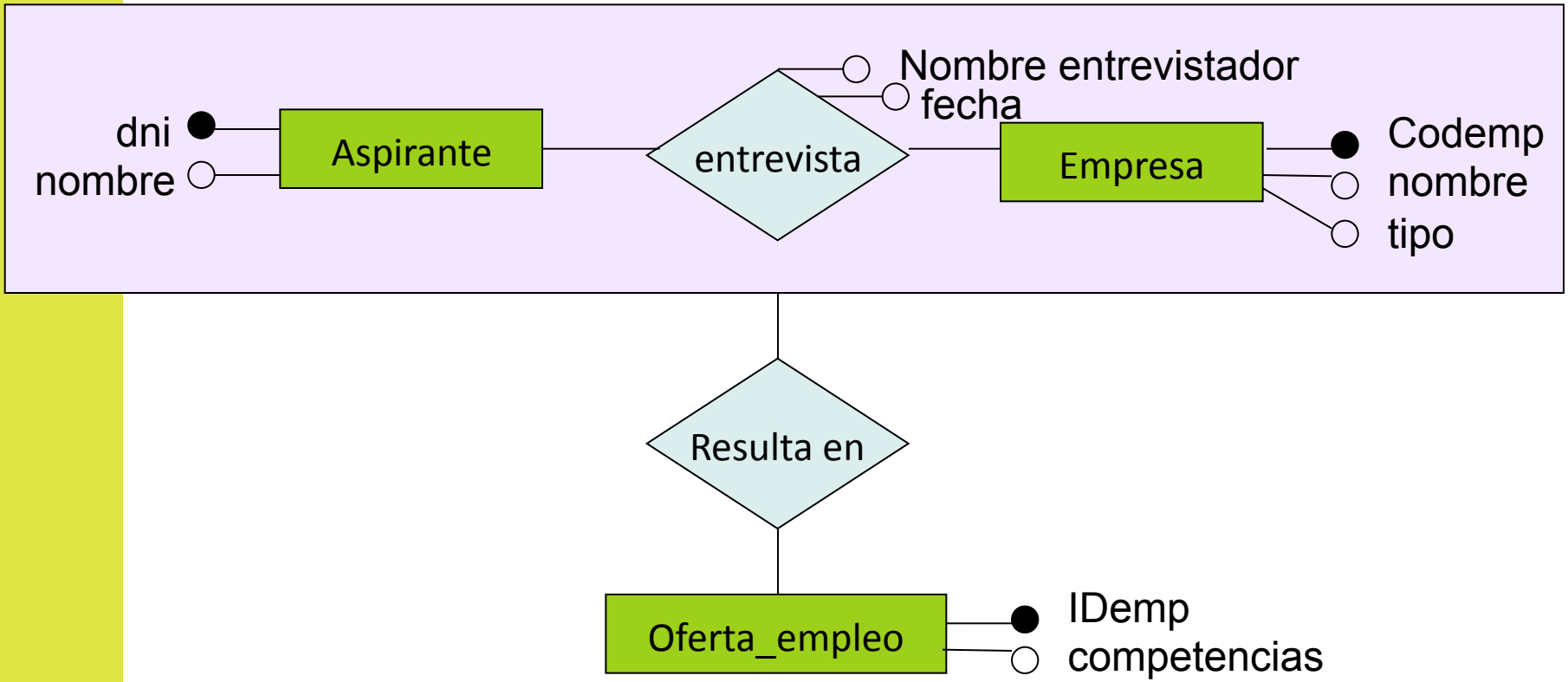


Ordenado por num_barco

Agregación
Miembro/Colección
con cardinalidades y
restricción de orden

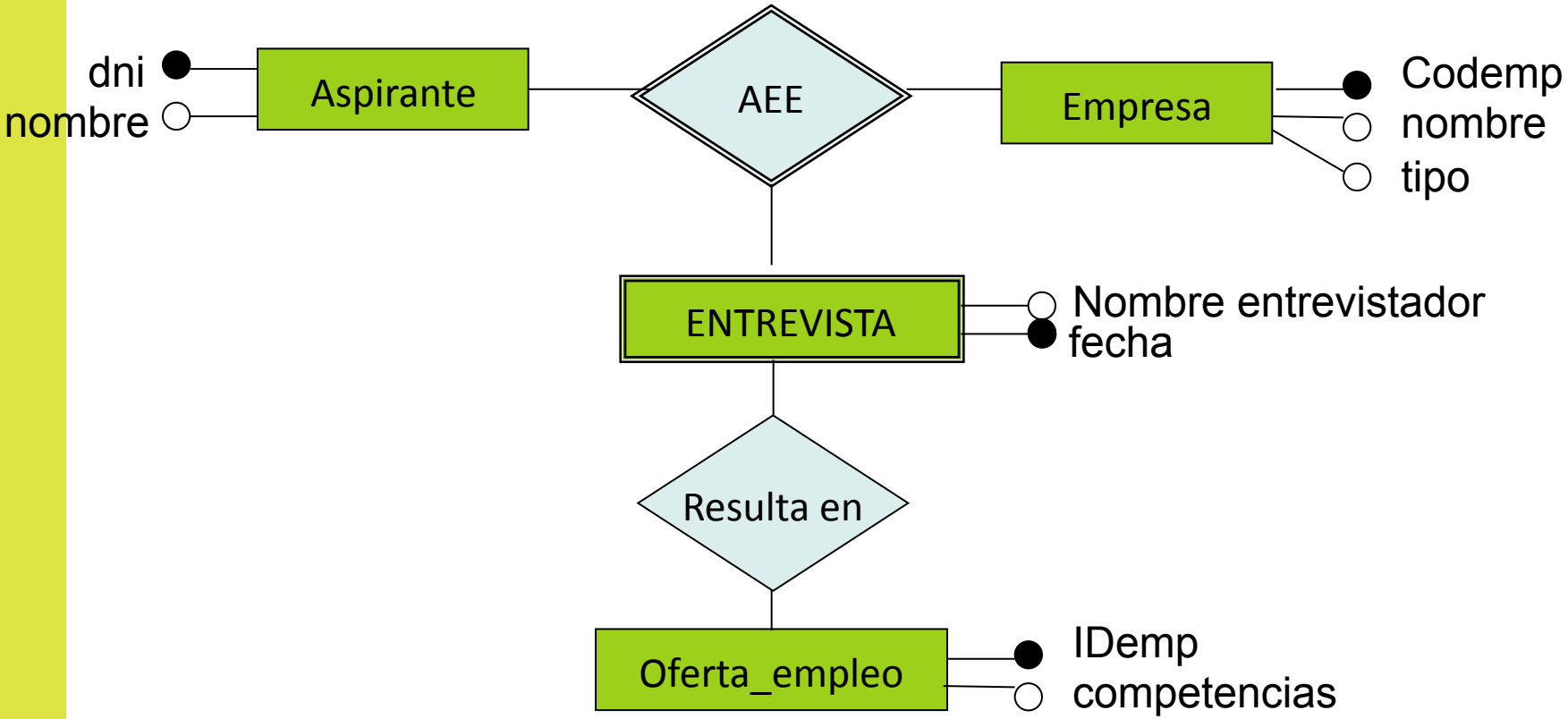
AGREGACIÓN – OTROS USOS

Como herramienta para expresar relaciones entre relaciones o entre relaciones y conjuntos de entidades



AGREGACIÓN – OTROS USOS (Y 2)

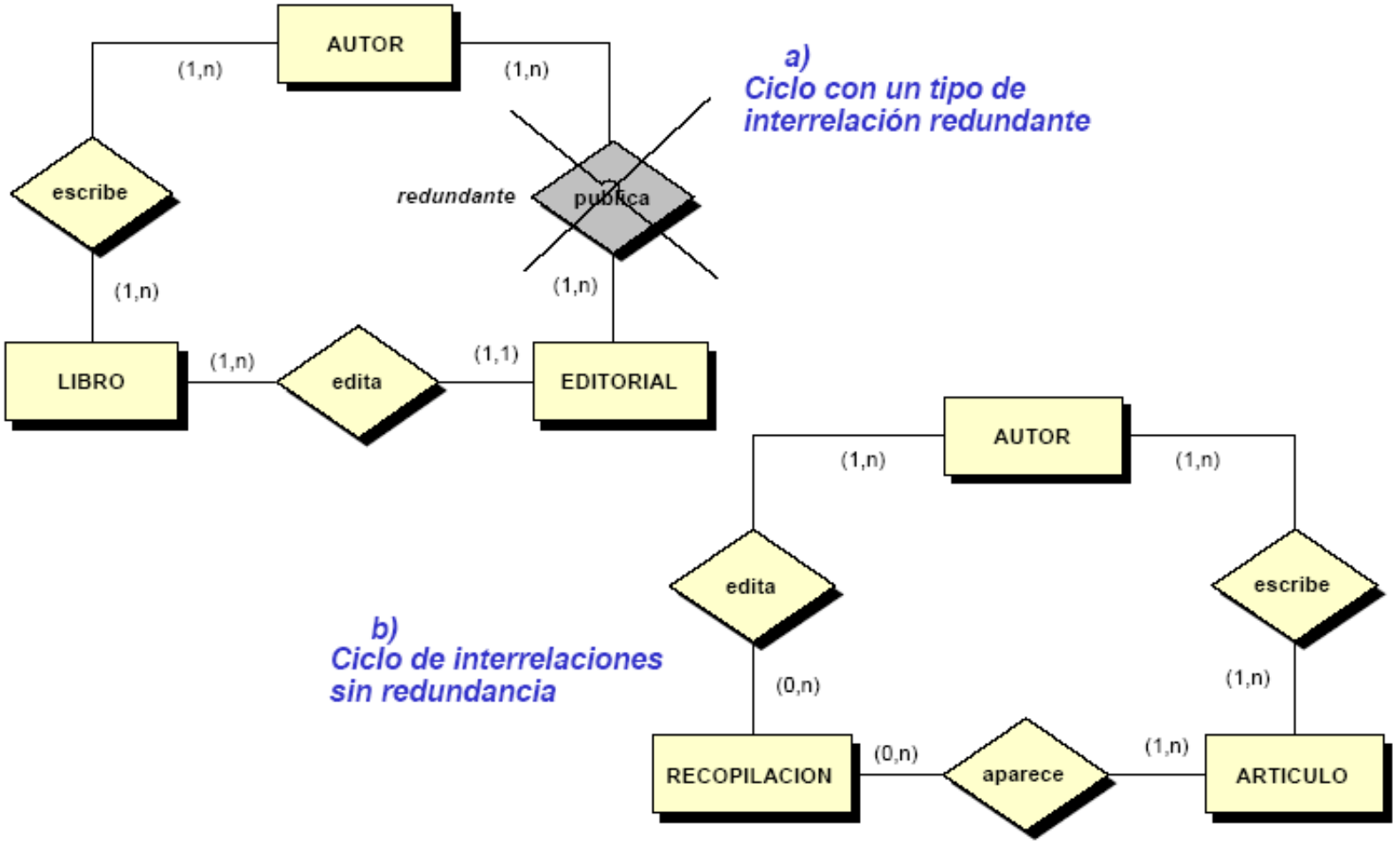
Una forma de resolver esta situación en ER es creando la entidad débil Entrevista y relacionar ésta con Oferta_empleo



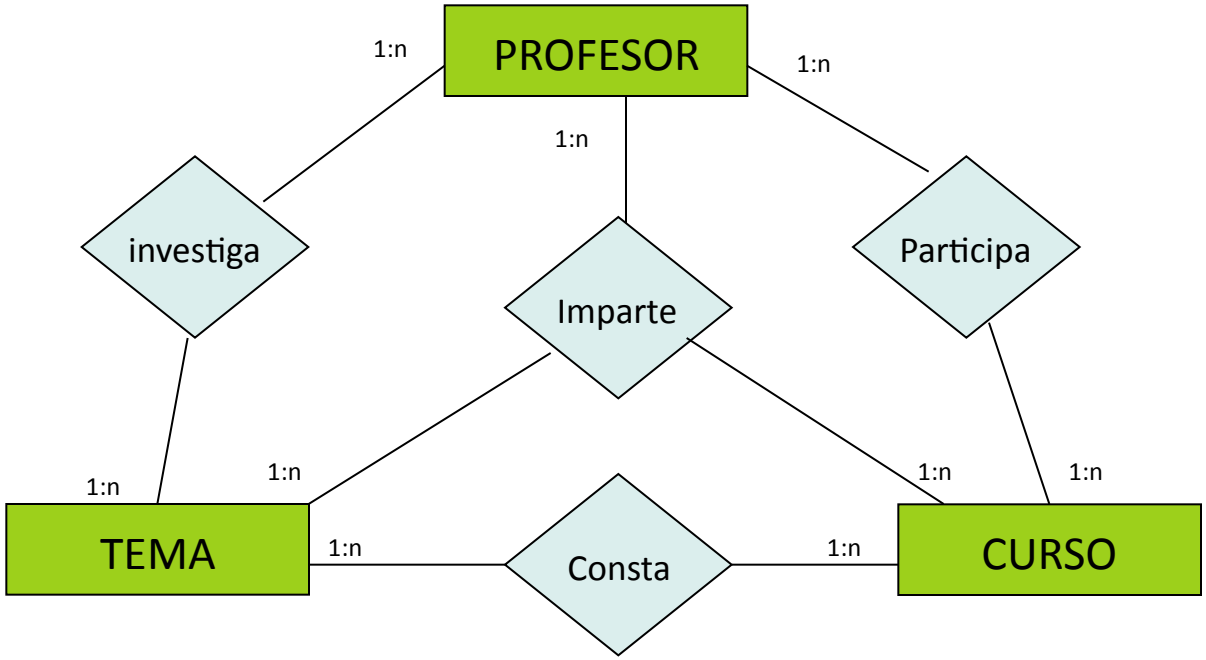
Evitar las redundancias

- Un elemento de un esquema es redundante si puede ser eliminado sin pérdida de semántica.
- Existen dos formas principales de redundancia:
  - En los atributos (derivados o calculados):
    - Aunque son redundantes, no dan lugar a inconsistencias siempre que en el esquema se indique su condición de derivados y la fórmula mediante la que han de ser calculados.
  - En las relaciones (también llamadas interrelaciones derivadas):
    - Una relación es redundante si su eliminación no implica pérdida de semántica porque existe la posibilidad de realizar la misma asociación de ejemplares por medio de otras relaciones.
    - Para ello es condición necesaria pero no suficiente que forme parte de un ciclo => Hay que estudiar detenidamente los ciclos en el diagrama E/R.

EVITAR LAS REDUNDANCIAS (Y 2)



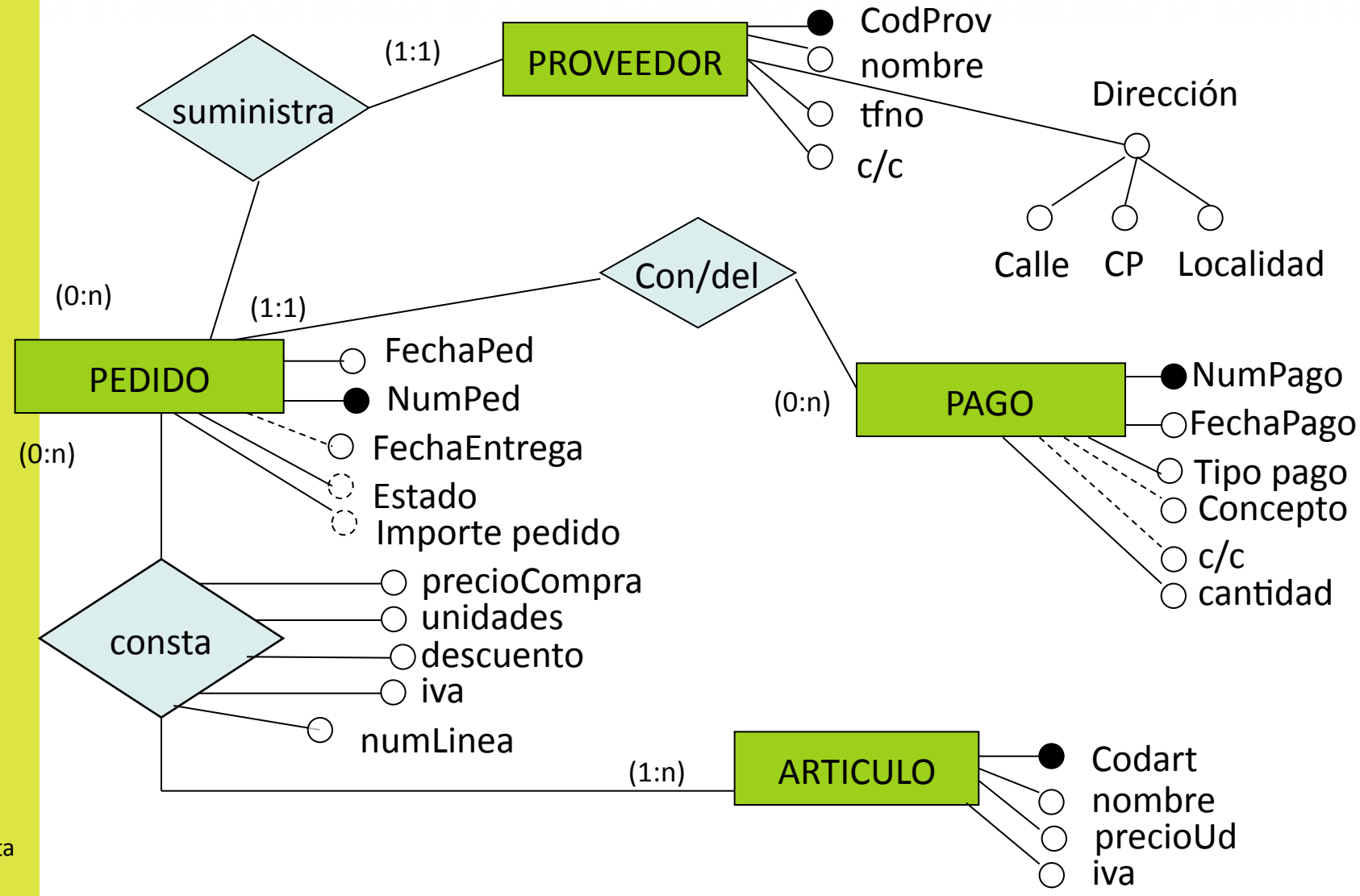
¿Hay problema de redundancia?



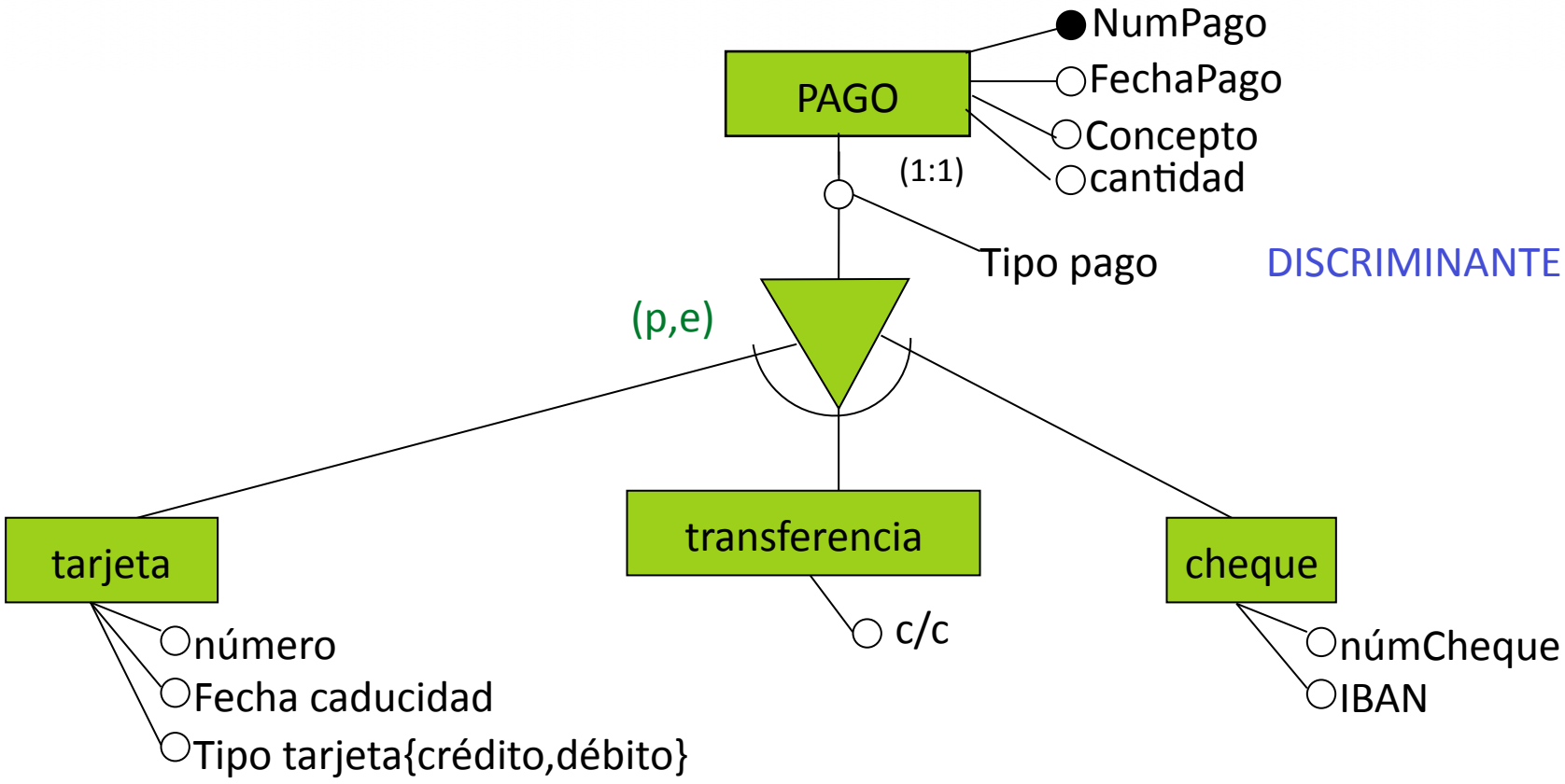
Gestión de compras (ejemplo)

- Requisitos:
  - Una empresa está interesada en automatizar su proceso de compras
  - El modo de funcionar es sencillo, básicamente requiere registrar la hoja del pedido que realiza a un determinado proveedor en una determinada fecha
  - En la hoja del pedido queda constancia del número de unidades que compra de cada artículo y el precio de compra, y en caso de que el proveedor o bien por volumen o por promoción, le realiza un descuento, también lo anota
  - Los productos que compran tienen distinto IVA
  - Generalmente el paga a sus proveedores al mes de recibir la mercancía y por transferencia, aunque lo puede hacer a plazos
  - Los atributos de cada entidad son los que cabría esperar

GESTIÓN DE COMPRAS



Generalización del tipo de pago

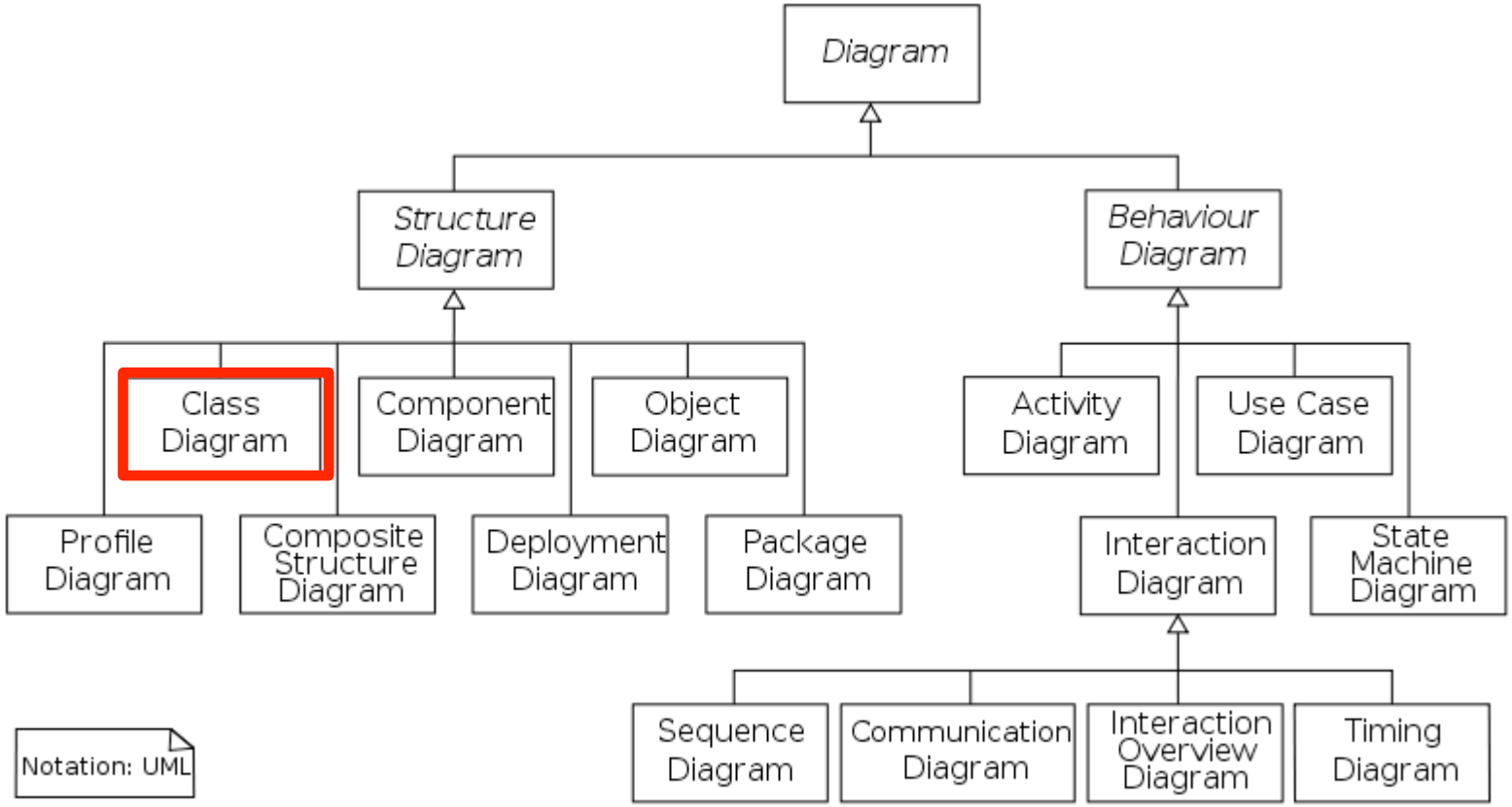


UML para diseño de BD

Diagramas UML

- UML se define como “un lenguaje estándar para la especificación, construcción, visualización y documentación de los componentes de un sistema software”
- Es independiente de un lenguaje de programación o proceso de desarrollo concreto
- Cubre la definición de aspectos **estáticos (estructurales) como dinámicos (de comportamiento)** del sistema mientras evoluciona a través del ciclo de vida del desarrollo haciendo uso de distintas vistas arquitecturales (Casos de Uso, de diseño, de Interacción, de Implementación, de Despliegue) y diagramas (casos de uso, de clases, de o objetos, de Secuencia, de Colaboración, etc.)
- Ha sido aceptado para diseñar BD

JERARQUÍA DE DIAGRAMAS UML



Diagramas UML

- **Diagramas de Estructura**
  - **Clases**: clases, relaciones, interfaces y colaboraciones. Vista de diseño estática y de procesos.
  - **Objetos**: modelan instancias de la clase y se utilizan para describir el sistema en un instante concreto de tiempo.
  - **Diagramas de componentes**: organización y dependencias entre un conjunto de componentes software (fuente, binarios y ejecutables).
  - **Diagramas de despliegue**: Muestran la configuración del sistema en tiempo de ejecución, indicando nodos hardware, componentes que se ejecutan en esos nodos, etc.
  - **Componentes**: Describen la estructura del software mostrando la organización y las dependencias entre un conjunto de componentes
  - **Estructura compuesta**: Muestran la estructura interna (incluyendo partes y conectores) de un clasificador estructurado o una colaboración

Diagramas UML

- **Diagramas de Comportamiento**
  - **Caso de uso**: modelan la funcionalidad del sistema, con los actores que intervienen
  - **Interacción**: objetos, relaciones y mensajes:
    - **Secuencia**: ordenación temporal de mensajes que intercambian los objetos de un caso de uso
    - **Colaboración**: muestra las interacciones entre objetos en forma de series de mensajes secuenciados
    - **Tiempo:** Muestran los tiempos reales en la interacción entre diferentes objetos o roles.
    - **De Revisión de Interacciones**: Híbrido entre diagrama de actividad y diagrama de secuencia.
  - **Estado**: muestran cómo pueden cambiar los objetos en respuesta a los sucesos externos. En general modelan las transiciones de un objeto específico.
  - **Actividad**: modelan el flujo de control de una actividad, representan la invocación de una operación, un paso dentro de un proceso de negocio o un proceso de negocio completo.

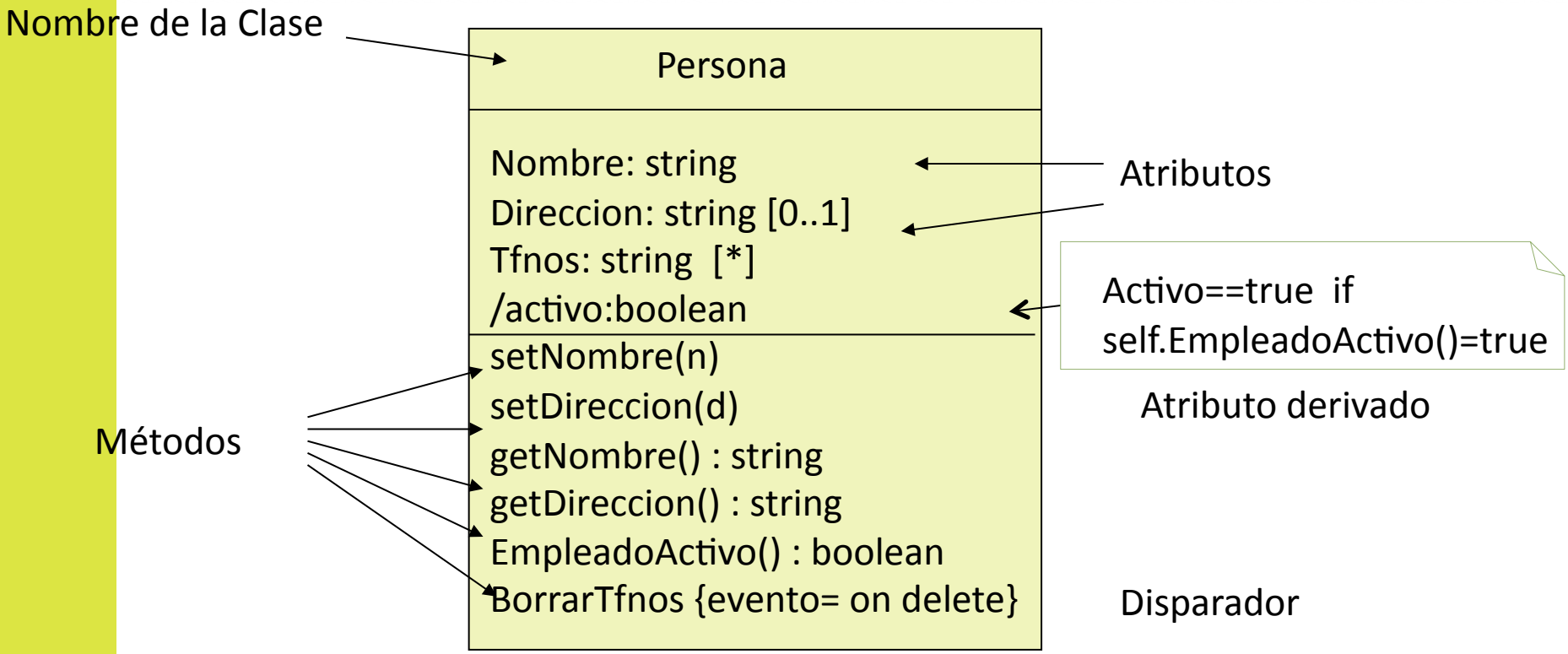
Diagrama de Clases

- **Diagrama de Clases:** muestra un conjunto de clases, interfaces y colaboraciones, así como las relaciones entre ellos.
- La **clase es la unidad básica que encapsula toda la** información de un Tipo de Objeto (un **objeto es una** instancia de una clase).
- Una **relación es una conexión entre elementos** estructurales.
- Existen cuatro tipos de **relaciones en UML2:**
  - Dependencia
  - **Asociación**
    - **Agregación**
    - **Composición**
  - **Generalización**
  - Realización

Clases

- Clase: Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica
  - Pueden ser concretas o abstractas
- Atributo: Representa una propiedad de una entidad. Cada atributo de un objeto tiene un valor que pertenece a un dominio de valores determinado.
  - Las sintaxis de una atributo es:
    - Visibilidad <nombre>: tipo = valor { propiedades}
  - Donde visibilidad es uno de los siguientes:
    - + público.
    - # protegido.
    - - privado.
- Operación: el conjunto de operaciones que describen el comportamiento de los objetos de una clase.
  - La sintaxis de una operación en UML es:
    - *Visibilidad nombre (lista_parámetros): tipo_retorno { propiedades}*

Ejemplo: Clase Persona



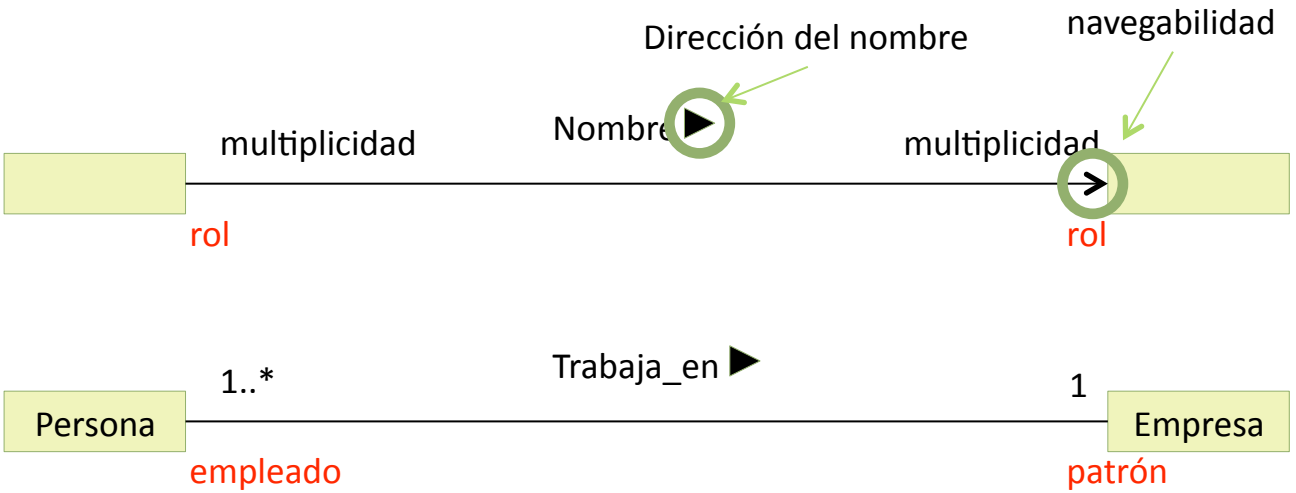
Nota: en BD, en general, no se indica la visibilidad pues se entiende siempre pública. Los métodos set y get se entienden siempre establecidos.

Asociaciones

- Las asociaciones representan relaciones existentes entre clases.
- Pueden ser:
  - Binarias
  - Reflexivas
  - Ternarias

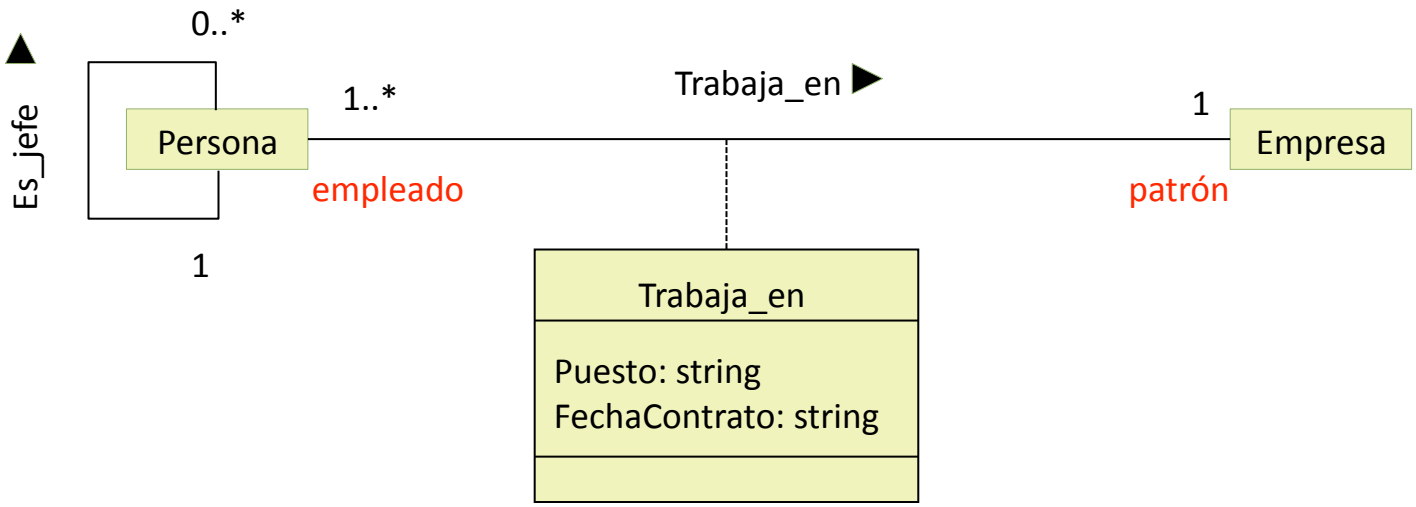
Multiplicidad:

1	*
0..1	n..m
0..*	



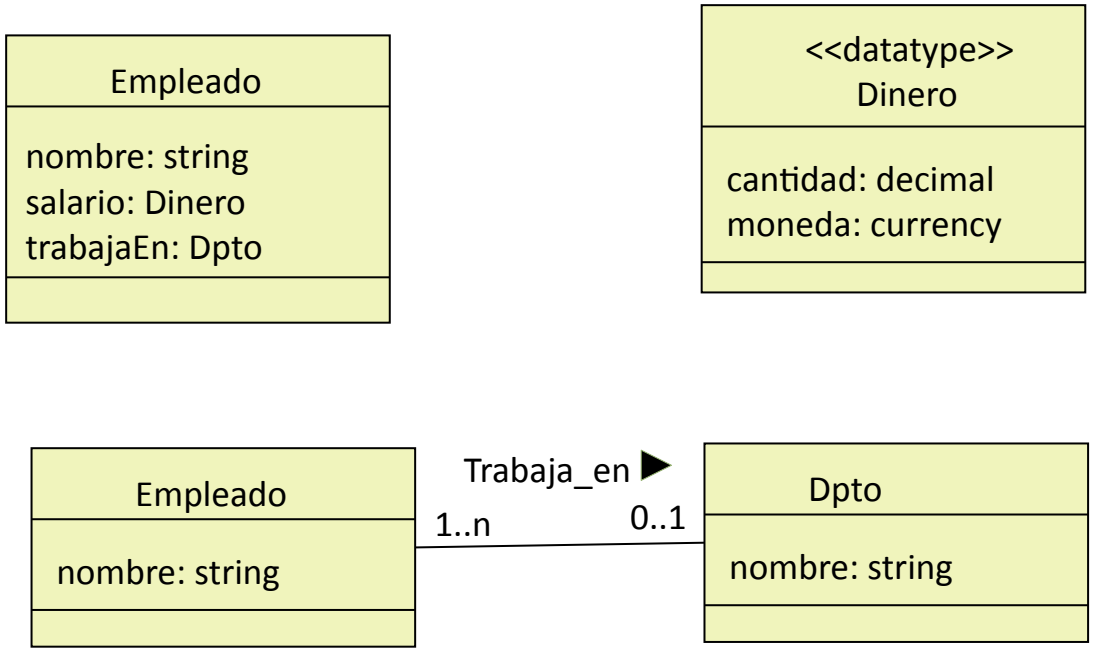
Asociaciones

- La asociación puede tener atributos
- Pueden ser reflexivas

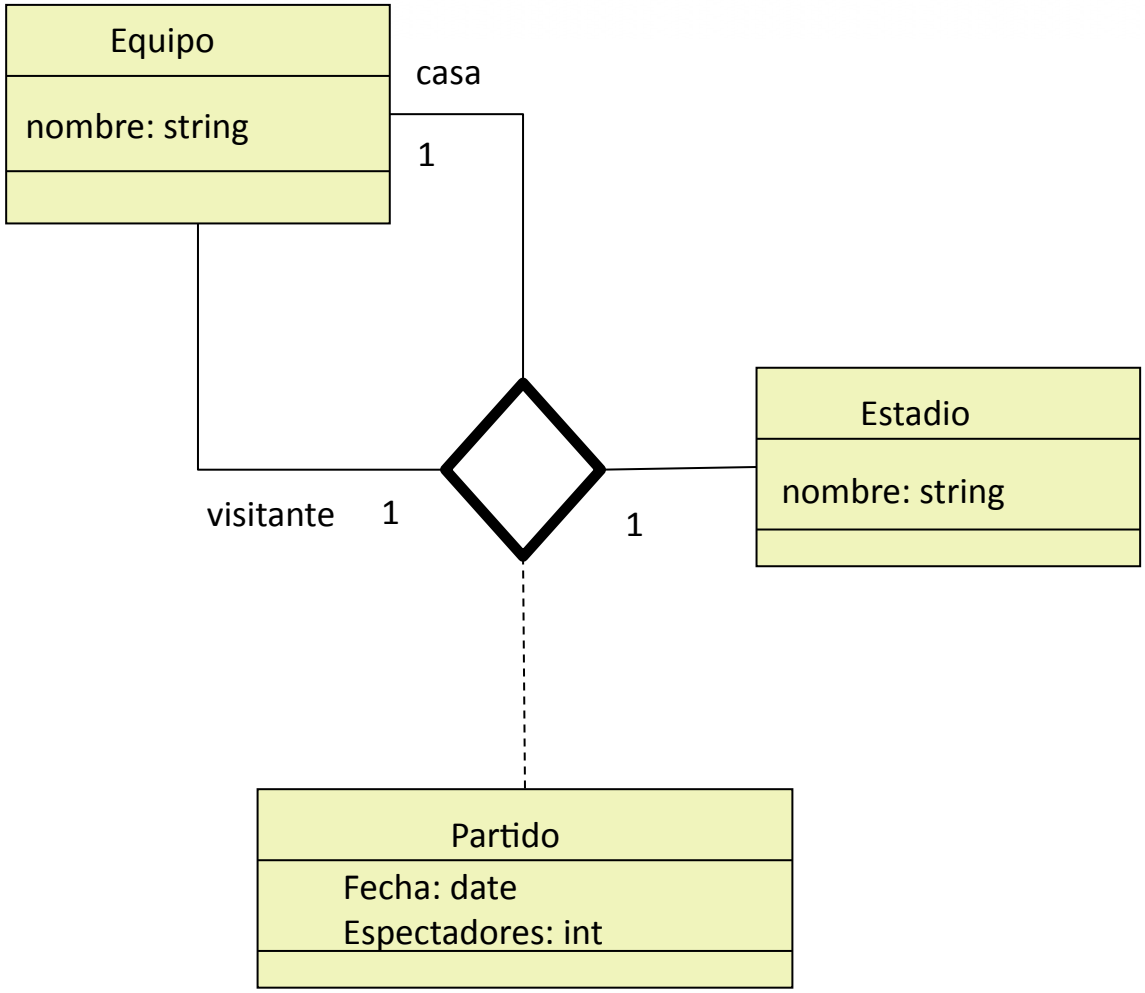


Asociaciones binarias como atributos

- En general, las relaciones binarias se establecen como asociaciones para una mejor interpretación
- Se aconseja modelar solo aquellos elementos específicos del dominio

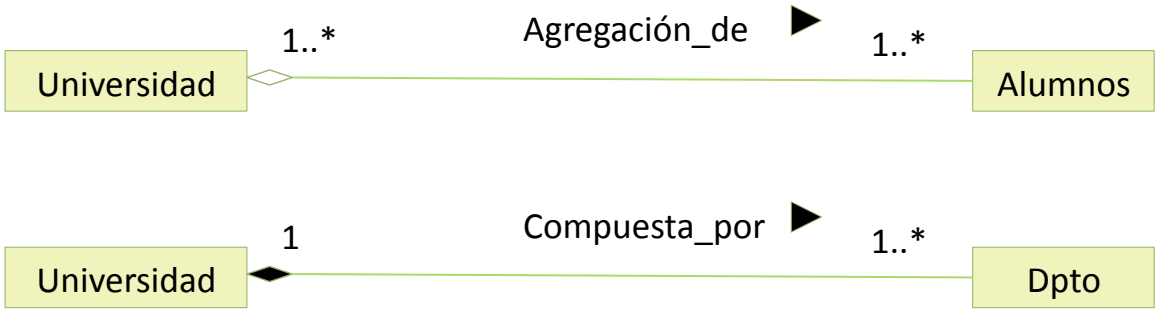


ASOCIACIONES N-ARIAS



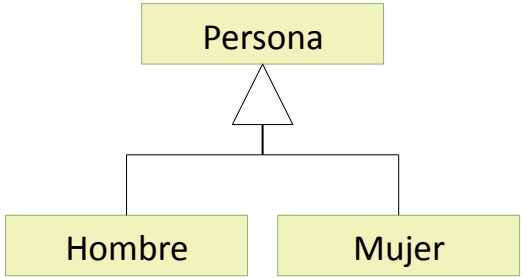
Agregación y Composición

- Una **agregación** es una asociación que permite representar objetos compuestos.
  - Agregación: cuando ambas clases existen de forma independiente
  - Composición: la existencia de una clase está condicionada a la existencia de otra**.**



Generalización

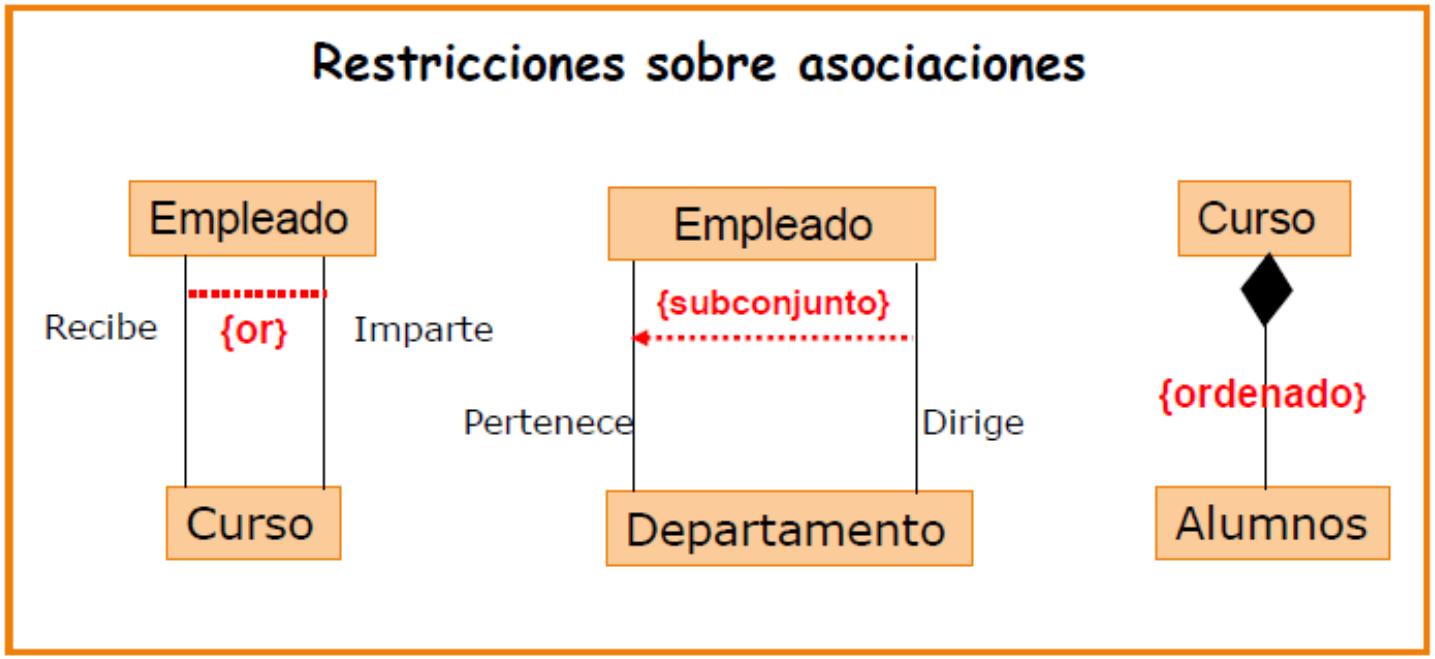
- La generalización es una asociación entre una clase más general (superclase o clase padre) y una clase más específica (subclase o clase hija).
- Lleva implícito dos principios: herencia (simple o múltiple) e inclusión



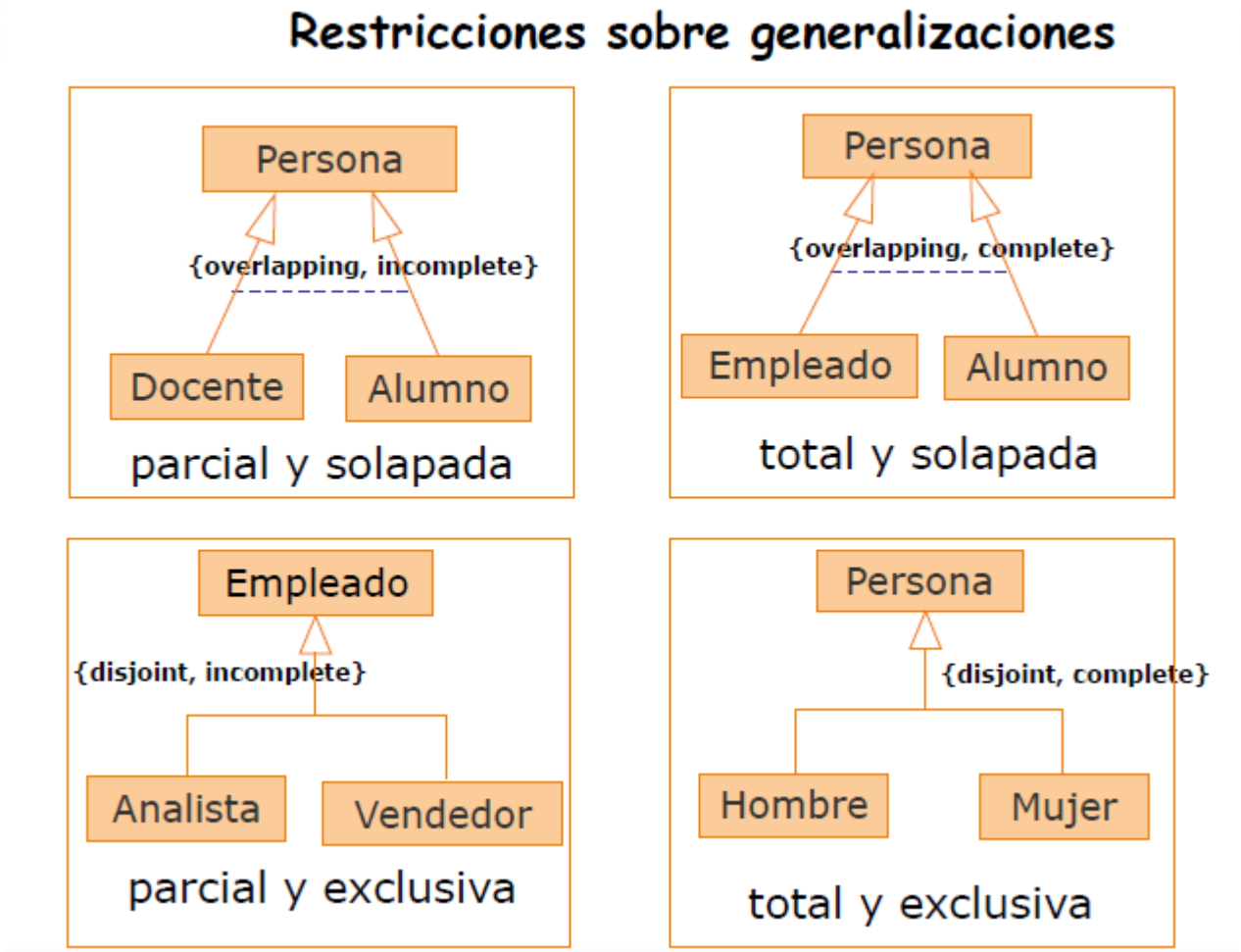
Mecanismos de extensión de UML

- Permiten adecuar la semántica de los elementos de modelos particulares
- *Extienden las posibilidades de anotación:*
  - Estereotipos: palabras claves que alteran el significado o funcionalidad de un elemento. <<persistent>>
  - Valor etiquetado: comentarios que permiten añadir información al elemento. {versión=3.1}
  - Restricción: limitan la funcionalidad de un elemento. {edad>18}
    - Se recoge en OCL

RESTRICCIONES



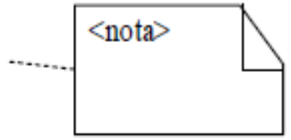
RESTRICCIONES

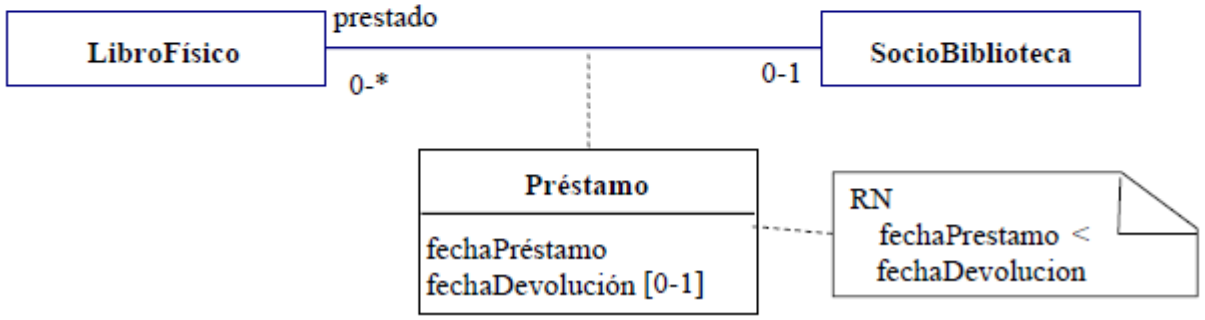


NOTAS

Nota:

es una observación, condición semántica o restricción





Restricciones en OCL

- OCL es un lenguaje de modelos y no lenguaje de programación
- Permite recoger la semántica de forma no ambigua
- Se utiliza para
  - Especificar invariantes para clases y tipos
  - Especificar pre- y post-condiciones para métodos
  - Como lenguaje de navegación
  - Especificar restricciones en operaciones
  - Comprobar requisitos y especificaciones

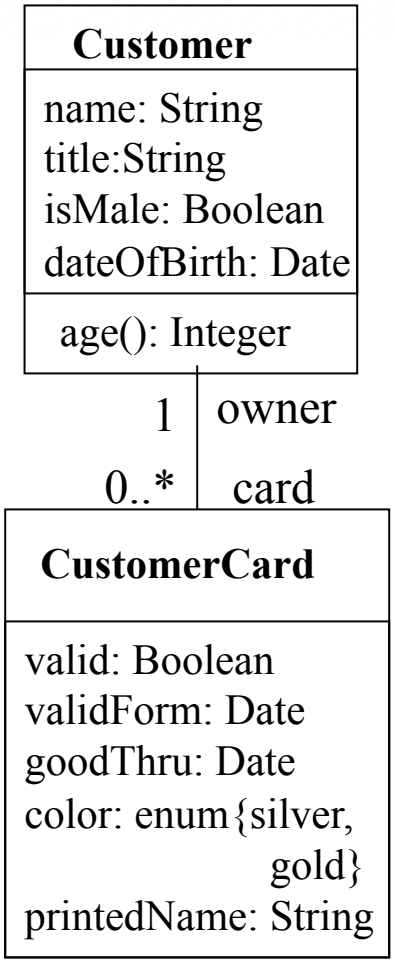
Restricciones en OCL: ejemplos

- Invariantes en atributos:

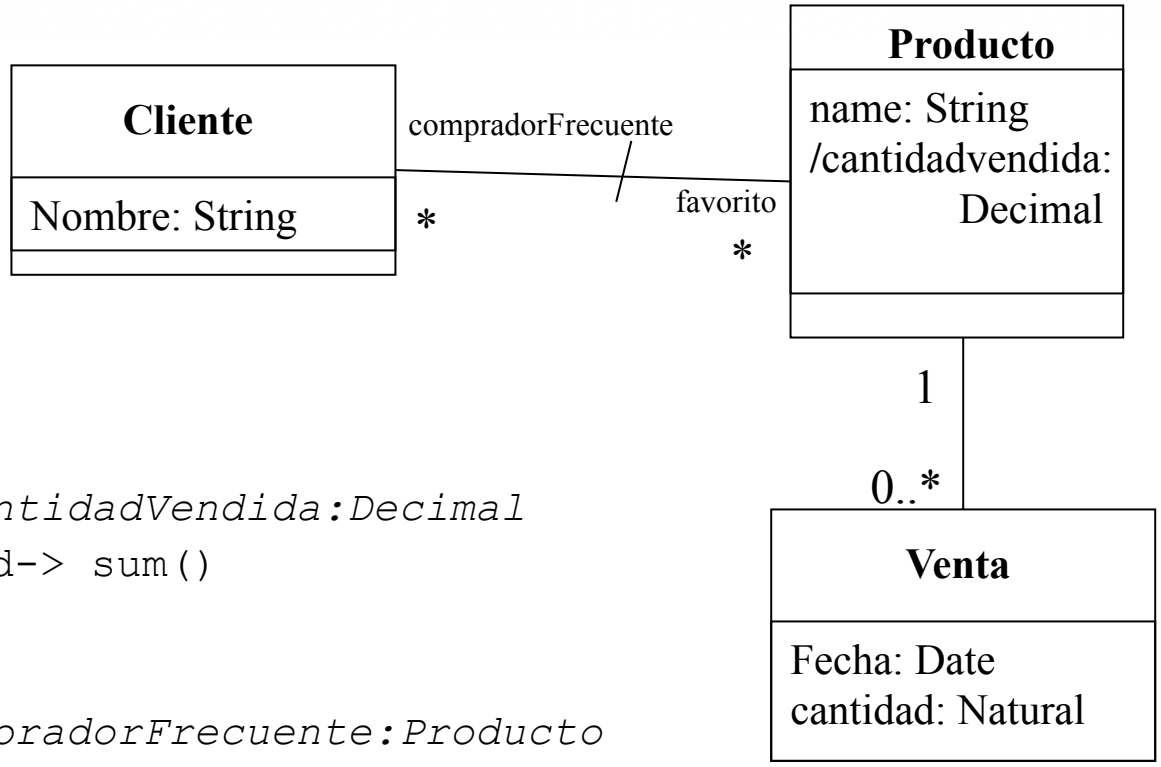
```
context Customer
invariant agerestriction: age >= 18

context CustomerCard
invariant correctDates:
    validFrom.isBefore(goodThru)

The type of validFrom and goodThru is Date.
isBefore(Date):Boolean is a Date operation.
```

- Invariantes usando navegación

```
context CustomerCard
invariant printedName:
printedName =
owner.title.concat(' ').concat(owner.name)
```



Atributos y Tipos Derivados: ejemplos



- Reglas de derivación:

```
context Producto::cantidadVendida:Decimal
derive Venta.cantidad-> sum()
```

```
context Cliente::compradorFrecuente:Producto
derive Cliente.AllInstances()-> select(c:cliente|c.venta->
        (select (s:venta|s.producto=self).cantidad-> sum()>1000)
```

```
context Cliente::favorite:Producto
derive Producto.AllInstances()-> select(p:producto|p.venta->
        (select (s:venta|s.cliente=self).cantidad-> sum()>1000)
```

Diseño del Comportamiento

- El diseño estructural recoge los hechos del sistema de información. Estos cambian en el tiempo de forma organizada (no arbitraria)
- El esquema de comportamiento define principalmente las restricciones y efectos de los eventos. Estos pueden ser de dominio o de acción.
- Se definen mediante OCL a modo de pre-condiciones y post-condiciones o bien mediante lenguaje procedimental (pseudocódigo). En UML se representan como especializaciones del dominio de los eventos o de las acciones.

Diseño del Comportamiento

- Eventos de dominio: evento que produce un cambio en el dominio
  - Ej: en Pedidos, la acción cambiar el estado del pedido a “recibido” y actualizar la unidades recibidas de cada artículo
- Acciones: evento invocado por un usuario u otro evento. Este puede ser iniciado por un evento temporal
  - Ej: en Pedidos, el 30 de diciembre enviar bono a aquellos clientes que hayan gastado más de 1000€
  - Ej: nóminas, enviar transacciones al banco para pago del personal

EJEMPLO: CARRITO DE LA COMPRA VIRTUAL

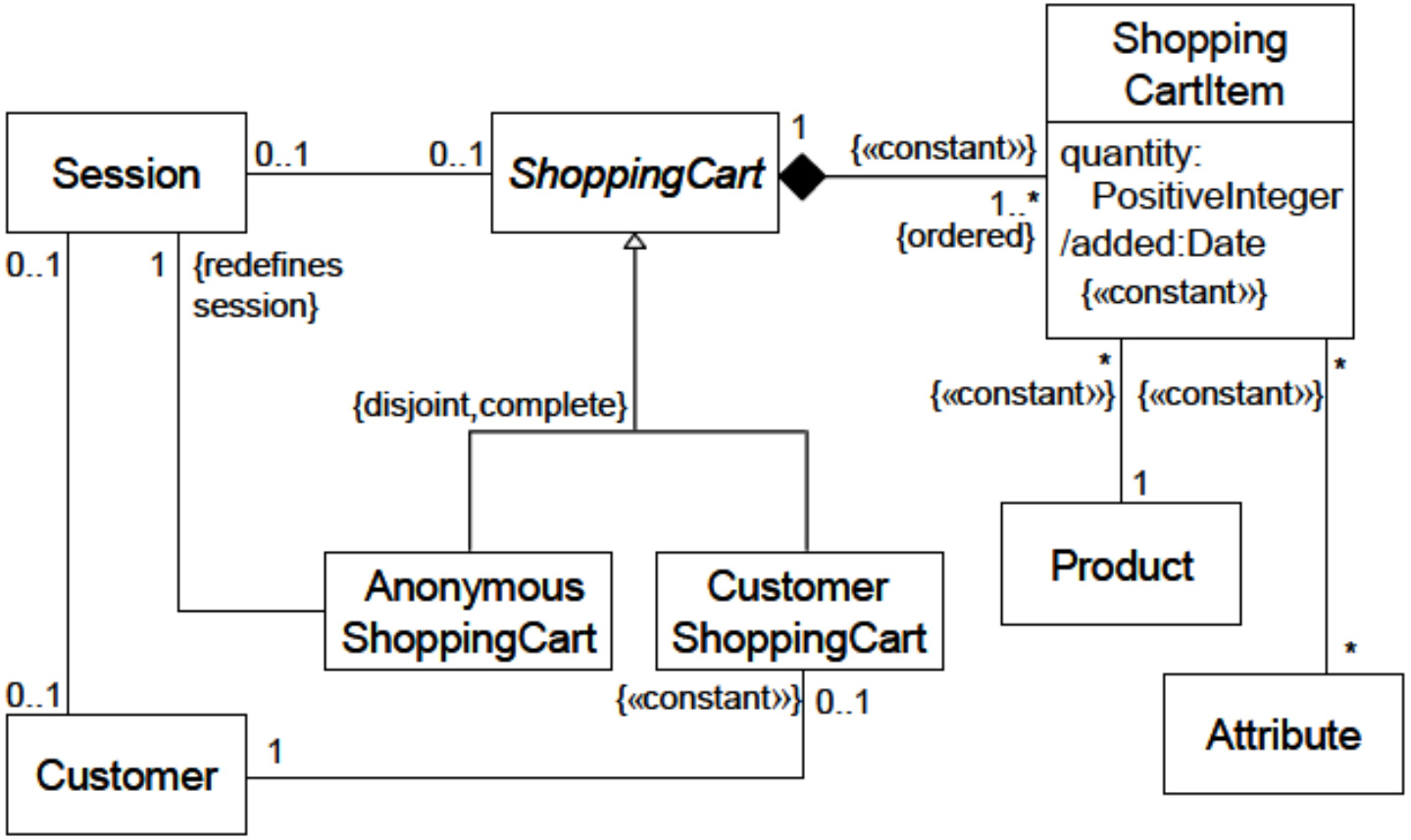


Fig. 16.15. Fragment of the structural schema dealing with shopping carts

(Olive et al, 2007)

EJEMPLO: CARRITO DE LA COMPRA VIRTUAL

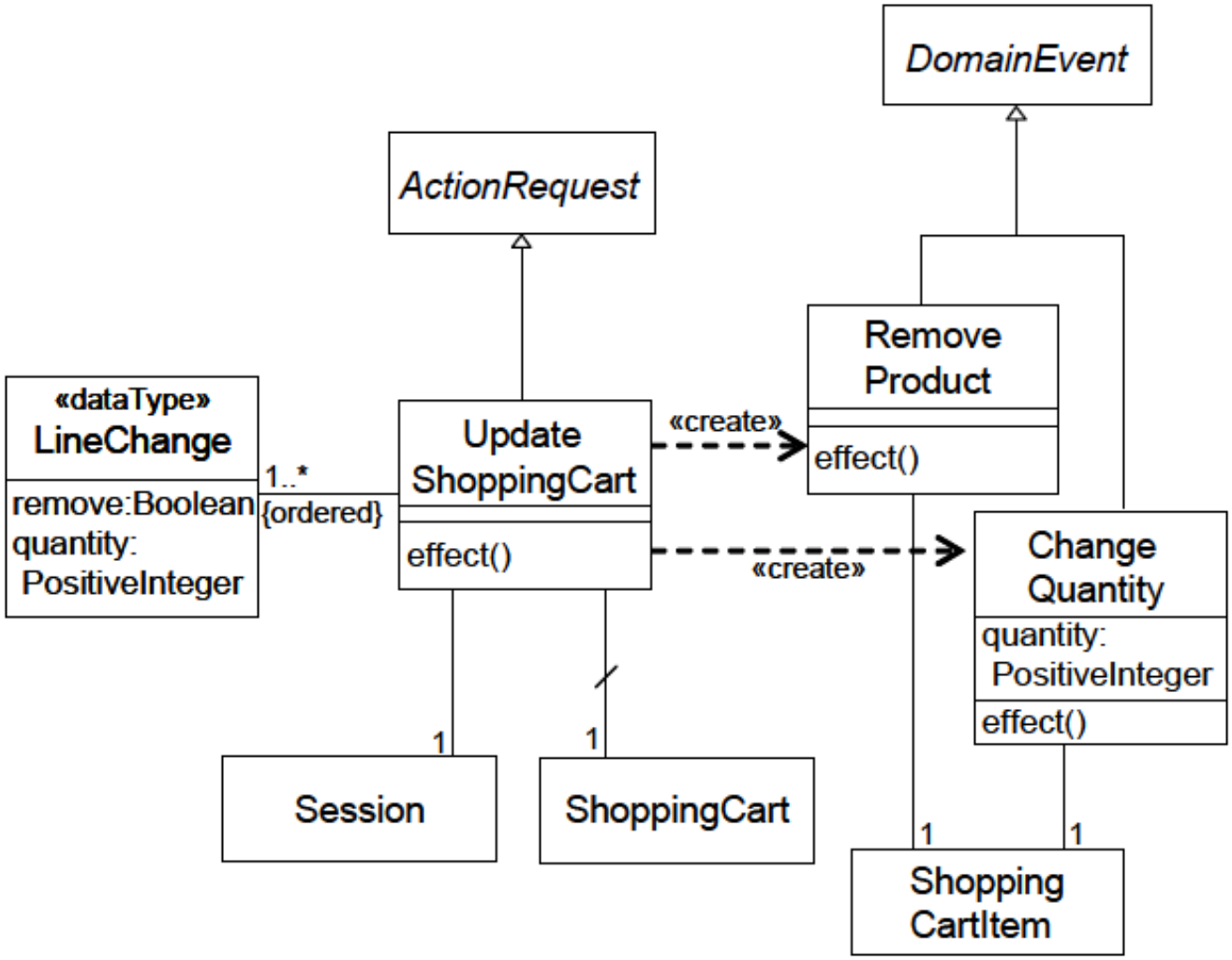


Fig. 16.16. Definition of the action request type *UpdateShoppingCart* and the domain event types *RemoveProduct* and *ChangeQuantity* (Olive et al, 2007)

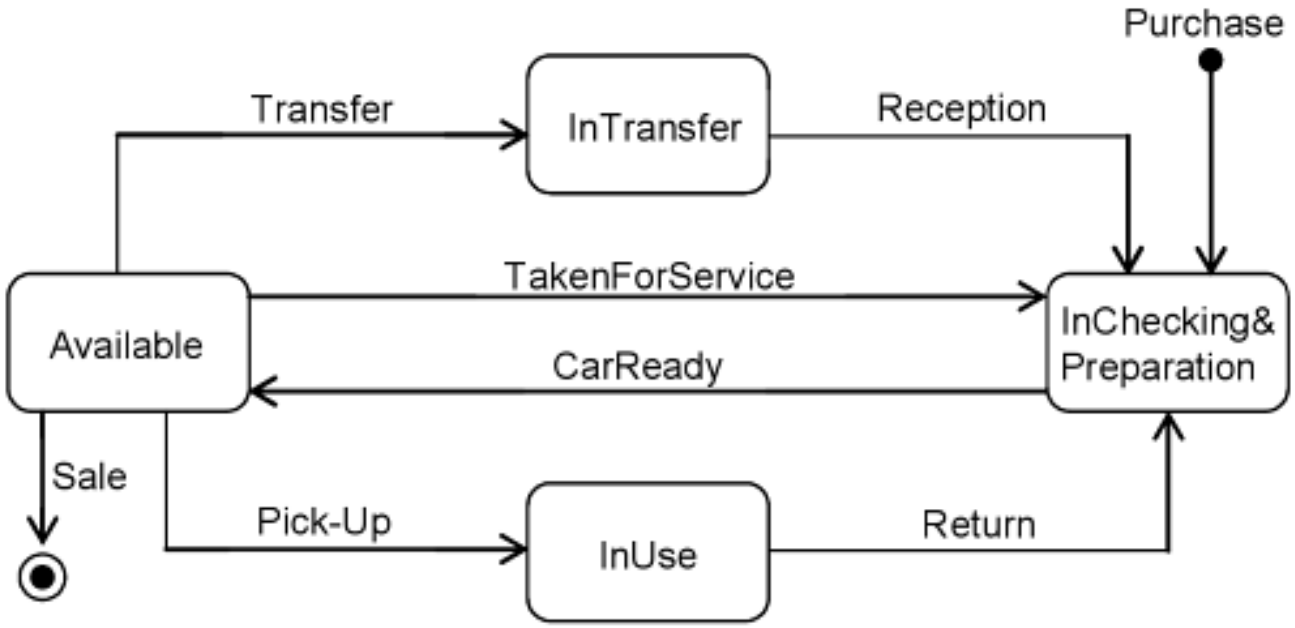
EJEMPLO: CARRITO DE LA COMPRA VIRTUAL

```
context RemoveProduct::effect()
  post: not shoppingCartItem@pre.oclIsKindOf(OclAny)


context ChangeQuantity::effect()
  post: shoppingCartItem.quantity = self.quantity


context UpdateShoppingCart::complete():Boolean
  body: lineChange->size() =
          shoppingCart.shoppingCartItem->size()

        context UpdateShoppingCart::effect()
          post:
            lineChange->forAll
              (lc|let cartItem:ShoppingCartItem =
                        shoppingCart.shoppingCartItem->
                          at(lineChange->indexOf(lc))
                   in
                     (lc.remove or lc.quantity <> cartItem.quantity)
                        implies
                          if lc.remove then
                             rp.oclIsNew() and
                             rp.oclIsTypeOf(RemoveProduct) and
                             rp.shoppingCartItem = cartItem
                          else
                             cq.oclIsNew() and
                             cq.oclIsTypeOf(ChangeQuantity) and
                             cq.shoppingCartItem = cartItem and
                             cq.quantity = quantity
                          endif))
```

(Olive et al, 2007)

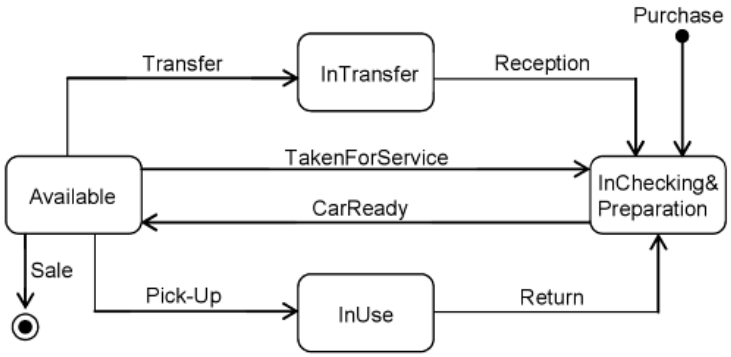
Diagrama de Transición de Estados

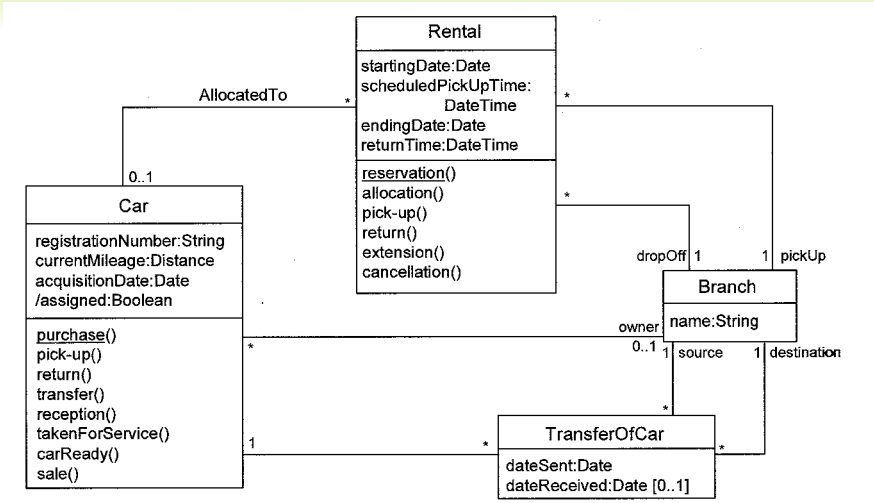
- Permiten recoger el comportamiento de un tipo de entidad
  - Ej: Estados por los que pasa un coche de alquiler



(Olive et al, 2007)

Diagrama de Transición de Estados






```
context Car::transfer(transferredTo:Branch)
  pre theCarIsFree: not assigned
  pre currentlyOwned: owner->notEmpty()
  post notOwned: owner->isEmpty()
  post transferCreated:
    t.oclIsNew() and
    t.oclIsTypeOf(TransferOfCar) and
    t.dateSent = CurrentDate and
    t.car = self and
    t.source = self.owner@pre and
    t.destination = transferredTo
```

```
context Car::reception(receivingBranch:Branch)
  post theBranchOwnsTheCar: owner = receivingBranch
  post transferCompleted:
    let pendingTransfer:TransferOfCar = transferOfCar
          ->select(dateReceived@pre->isEmpty())->any(true)
    in pendingTransfer.dateReceived = CurrentDate
```

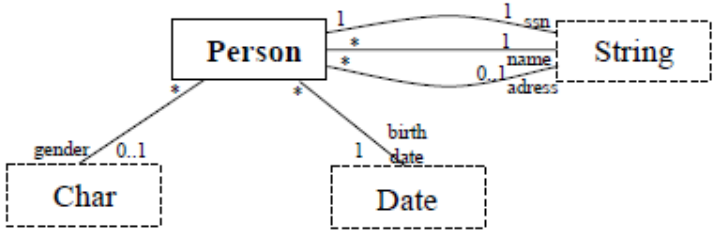
(Olive et al, 2007)

UML vs ER

UML vs ER

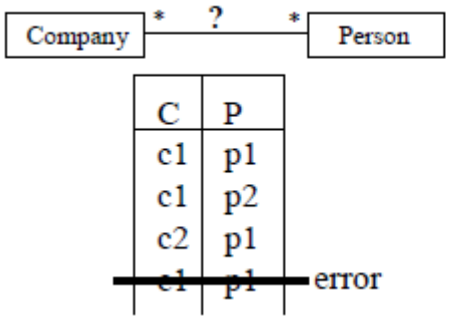
- ER no permite modelar el comportamiento (no hay operaciones ni mensajes)
- Los atributos en ER deberían considerarse en UML como asociaciones n..1 con la clase opuesta suprimida

Person	
ssn	: String
name	: String
adress	: String
gender	: Char
birthdate	: Eate



- ER representa la relación con un rombo, UML por medio de una asociación. No tiene la misma semática

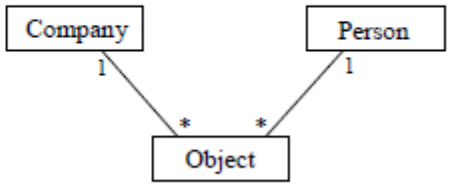
¿un emplado puede estar contratado varias veces por la misma empresa?



object-oriented "solution"

oid	C	P	
1	c1	p1	
2	c1	p2	
3	c2	p1	
4	c1	p1	ok

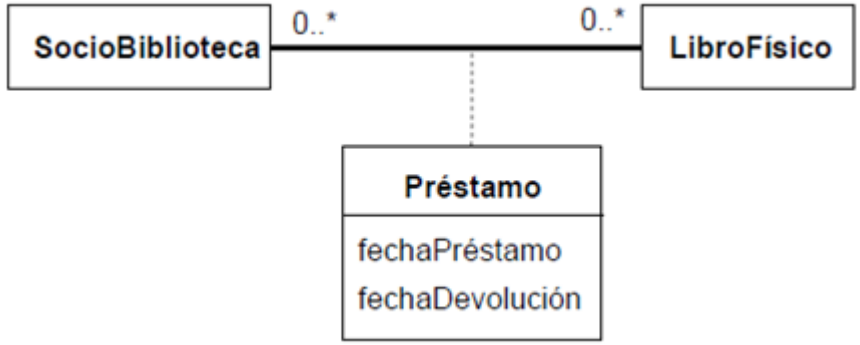
actual model



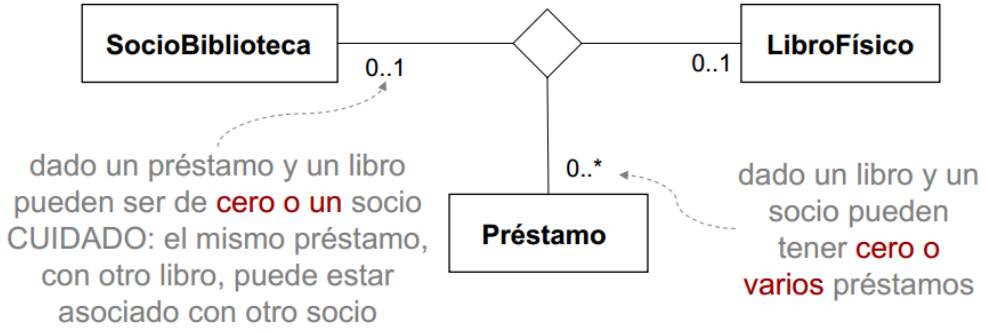
UML vs ER (y 2)

- En OO, cada objeto tiene un identificador único independientemente de sus atributos. ER utiliza la "key" para identificar cada entidad
- UML no permite recoger restricciones (más que la multiplicidad), se deben indicar en OCL
- En UML la relaciones ternarias no tienen el mismo significado que en ER

un préstamo es la asociación entre un único socio y un único libro (por la propia definición de clase asociativa)



Multiplicidad nos indica por cada pareja de instancias de dos clases, con cuántas instancias de la otra clase está relacionada
Pero, **NO** indica en cuántas asociaciones participa una instancia de una clase, hay que recogerlo en OCL



Pros y contras para diseño de BD con UML

- **Ventajas de UML**
  - Es **estándar => Facilita la comunicación**
  - Está basado en un **metamodelo con una semántica bien definida**
  - Se basa en una **notación gráfica concisa y fácil de aprender**
  - Se puede utilizar para modelar **sistemas software completos en diversos dominios:**
    - Sistemas de información empresariales, Sistemas WEB, sistemas críticos y de tiempo real, etc.
  - Es fácilmente extensible mediante perfiles

- **Inconvenientes de UML**
  - **No es específico para diseño de BD**
  - **No tiene la misma expresividad (semántica) que ER**
  - **No hay un perfil estandarizado, solo iniciativas (RationalRose, por ejemplo)**