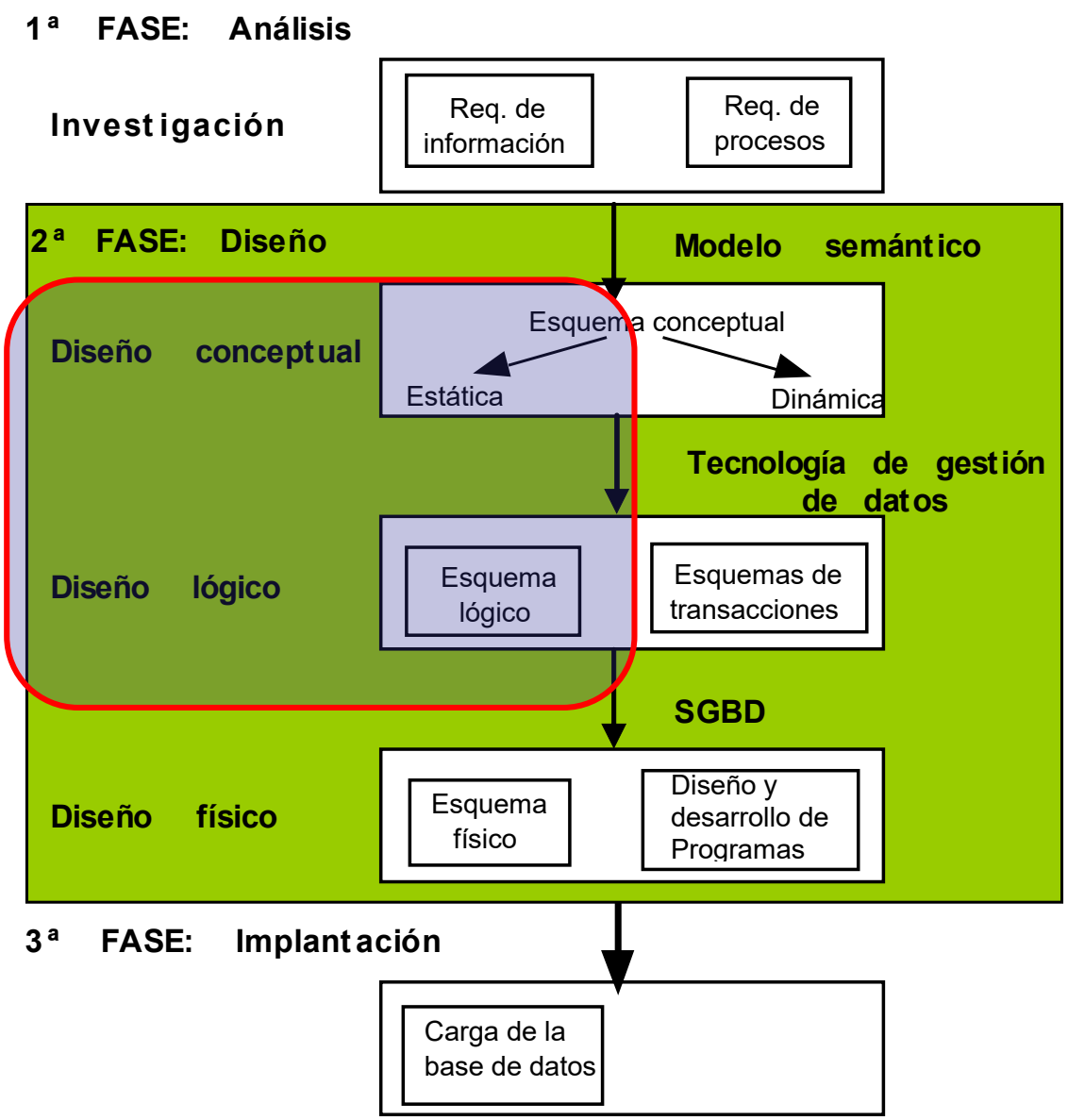
Unidad Didáctica 4:
Diseño de Bases de Datos Relacionales

Parte 2:
Diseño Conceptual
(Doc. UD4.2)

UD 4.2.- Diseño Conceptual

1.- Introducción



UD 4.2.- Diseño Conceptual

2.- Diseño Conceptual

Diseño conceptual: fase del diseño de una BD cuyo objetivo es “obtener una representación de la realidad que capture las propiedades estáticas y dinámicas de dicha realidad que son necesarias para satisfacer los requisitos; esta representación debe suponer una imagen fiel del comportamiento del mundo real”.

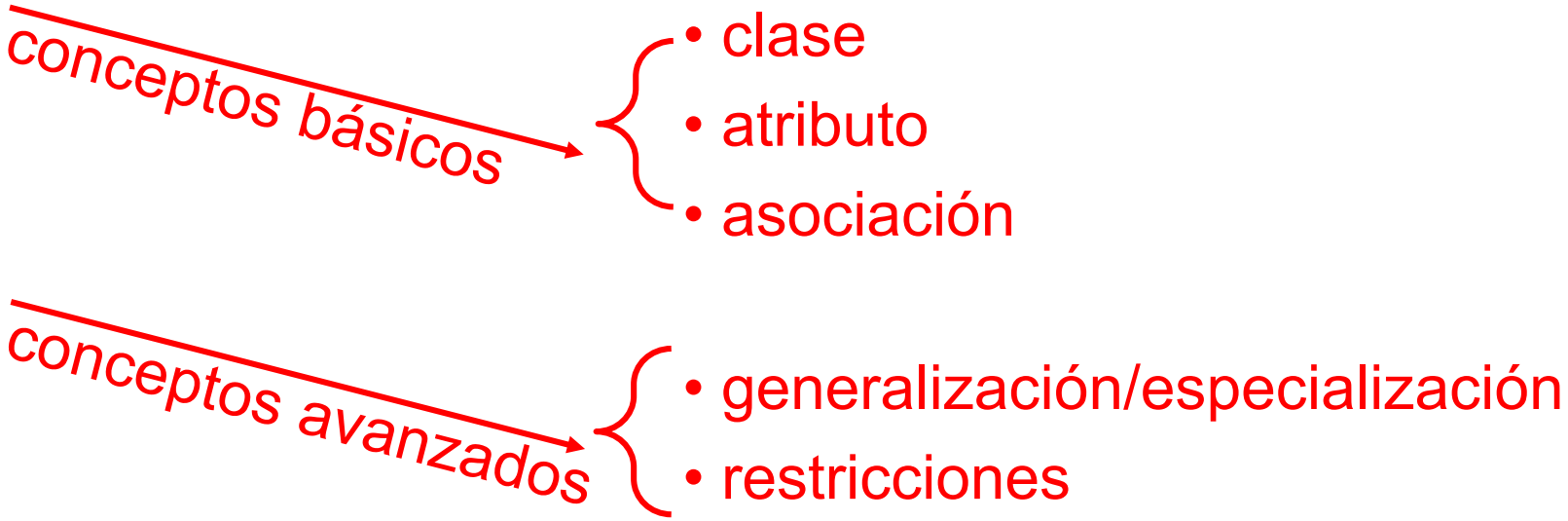
¿qué se va a usar para esa representación?

Diagrama de clases del UML

UD 4.2.- Diseño Conceptual

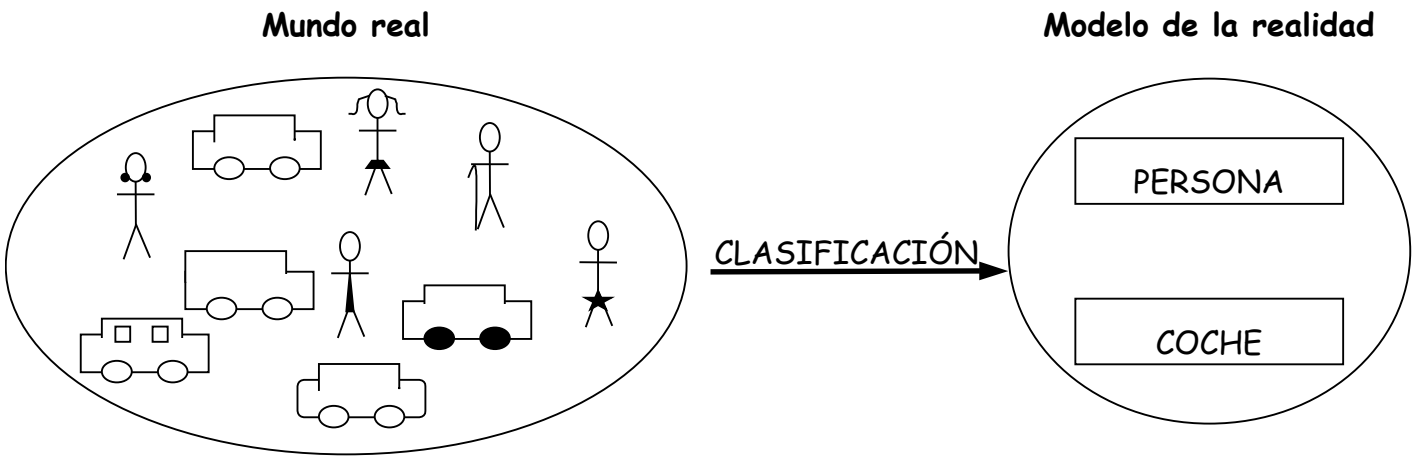
2.1.- El diagrama de clases del UML

El *diagrama de clases* permite representar las estructuras que constituyen el contenido del sistema de información junto con restricciones de distintos tipos que limitan las ocurrencias válidas de las mismas.



2.1.1.- Clase

La observación de la realidad permite detectar el conjunto de "objetos" (físicos o conceptuales) de los que se quiere almacenar información para, mediante el uso de la clasificación, que es uno de los mecanismos de abstracción más primario que existen, descubrir el conjunto de "clases" (o tipos de objetos) que son de interés para la organización

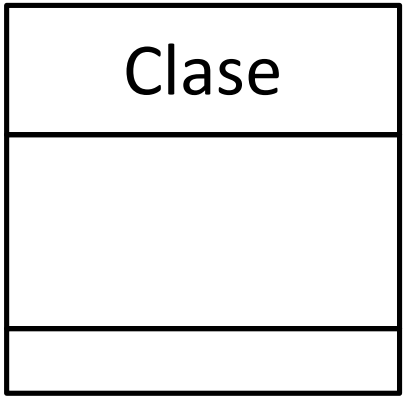


2.1.1.- Clase

- Los componentes básicos de un SI son los objetos de los que se quiere almacenar información; todos los objetos que son del mismo tipo se representan con un una clase.

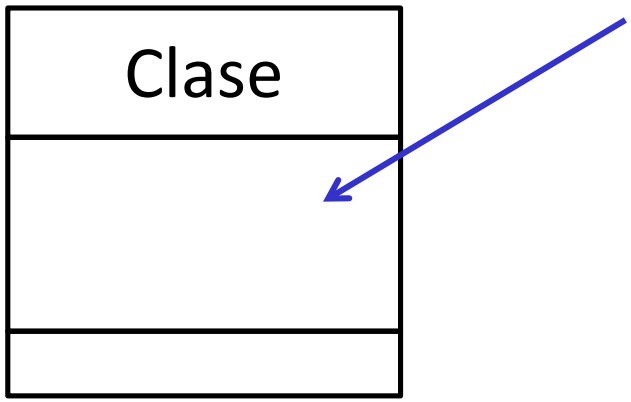
  Una clase es la descripción de un grupo de objetos con estructura, comportamiento y relaciones similares.

- Con una clase se representará cualquier persona, concepto, suceso o evento (en definitiva, cualquier “cosa”) sobre el que se quiera almacenar información.



2.1.2.- Atributo

Un atributo es una propiedad de una clase identificada con un nombre, cuyas instancias pueden tomar valor de un conjunto que se especifica.



Zona de especificación de los atributos de la clase

2.1.2.- Atributo

A cada atributo se le puede especificar:

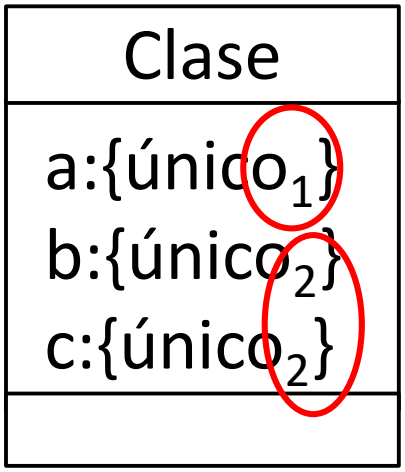
- Tipo de datos asociado que determina los valores que puede tomar el atributo:
  - El nombre de un tipo de datos conocido.
  - Una enumeración de valores posibles entre paréntesis.

- Restricciones:
  - de unicidad
  - de cardinalidad
  - de identificación

2.1.2.- Atributo

- Restricciones de unicidad:

Representa el hecho de que las distintas ocurrencias de una clase deben tomar valores distintos para el atributo (o conjunto de atributos) sobre los que se define



No puede haber dos ocurrencias de la clase con:

- el mismo valor en *a,* ni
- con el mismo valor a la vez en *b* y c

2.1.2.- Atributo

- Restricciones de cardinalidad:

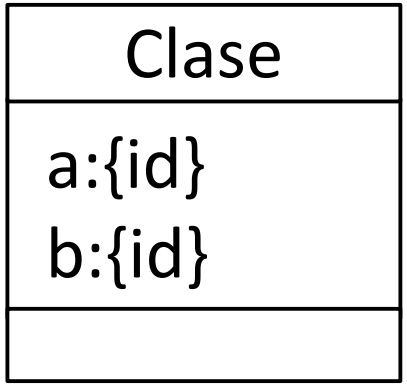
Expresan el número de valores que puede tomar el atributo para cada ocurrencia de la clase.

- {1..1}: el atributo tiene exactamente un valor para cada ocurrencia de la clase.
- {0..1}: el atributo puede tener como mucho un valor para cada ocurrencia de la clase o puede no tener valor (**es el valor por defecto**).
- {1..*}: el atributo puede tener más de un valor para cada ocurrencia de la clase pero al menos tiene un valor.
- {0..*}: el atributo puede tener más de un valor para cada ocurrencia de la clase o puede no tener valor.

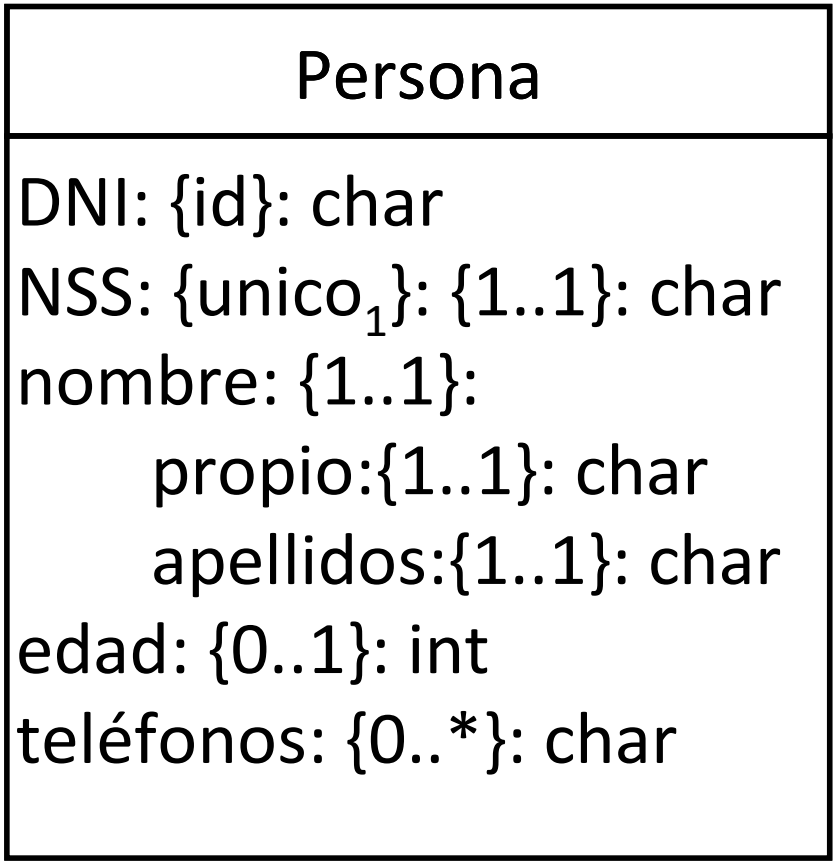
2.1.2.- Atributo

- Restricción de identificación:

Un identificador es un conjunto de atributos con restricción de unicidad y con cardinalidad {1..1} que permite distinguir dos ocurrencias de la clase (sólo hay un identificador aunque puede constar de varios atributos)



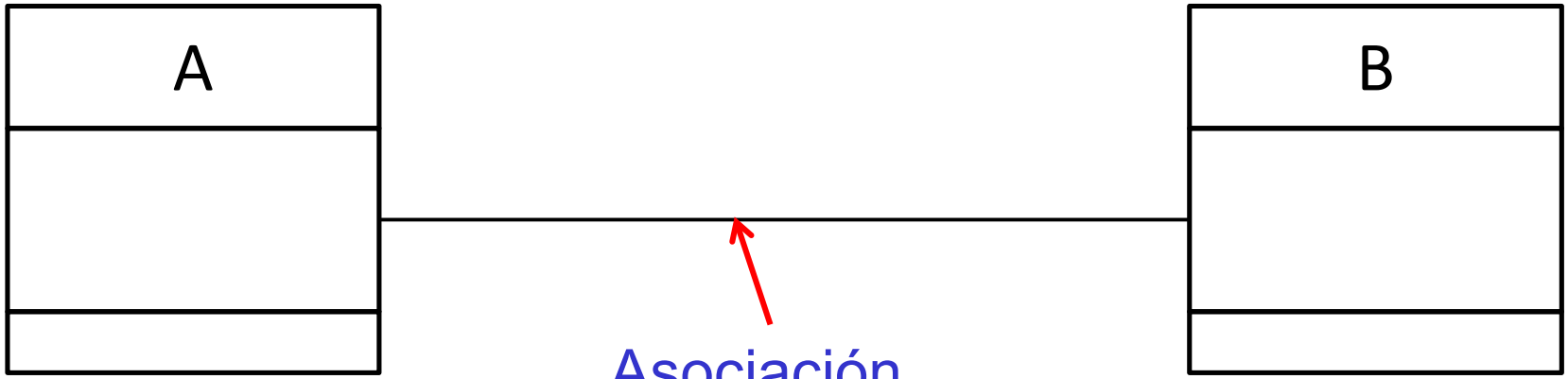
2.1.2.- Atributo



2.1.3.- Asociación

Las asociaciones permiten representar las posibles relaciones existentes entre las clases del sistema de información.

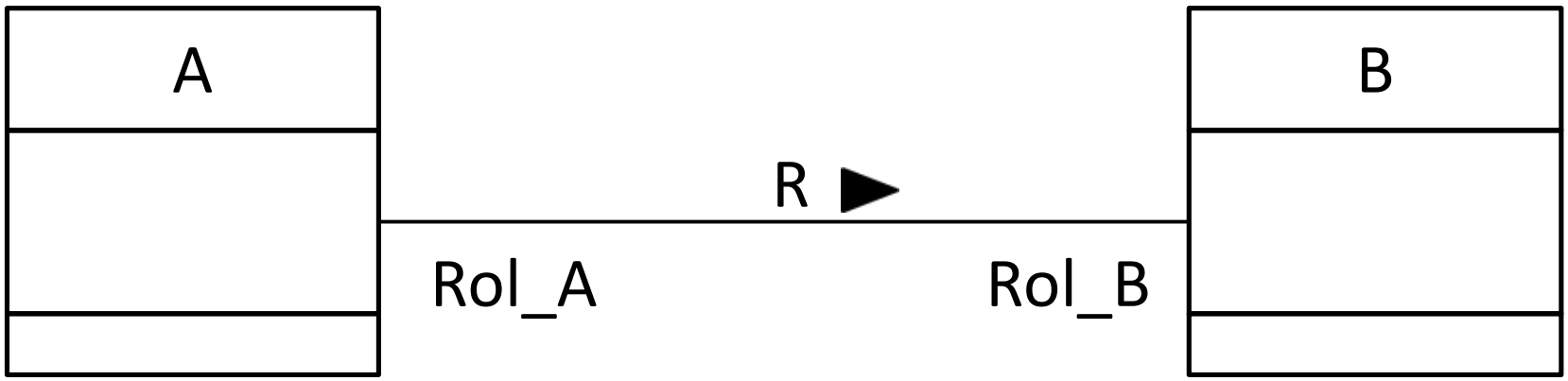
Una asociación que conecta exactamente dos clases se dice que es binaria.



2.1.3.- Asociación

"Adornos" en una asociación:

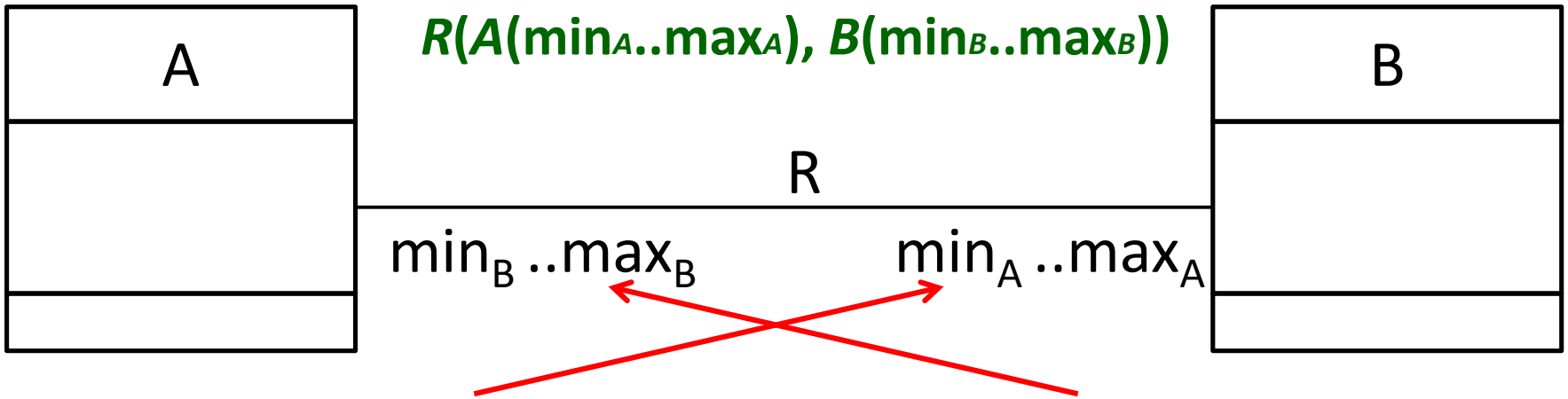
- Sentido (opcional, para clarificar la relación)
- Nombre
- Rol (opcional, para clarificar la relación)
- Cardinalidad



2.1.3.- Asociación

"Adornos" en una asociación:

- Sentido (opcional, para clarificar la relación)
- Nombre
- Rol (opcional, para clarificar la relación)
- Cardinalidad



Cada ocurrencia de *A* se relaciona con, como mínimo min_A ocurrencias de *B* y como máximo con max_A ocurrencias de *B*

Cada ocurrencia de *B* se relaciona con, como mínimo min_B ocurrencias de *A* y como máximo con max_B ocurrencias de *A*

2.1.3.- Asociación

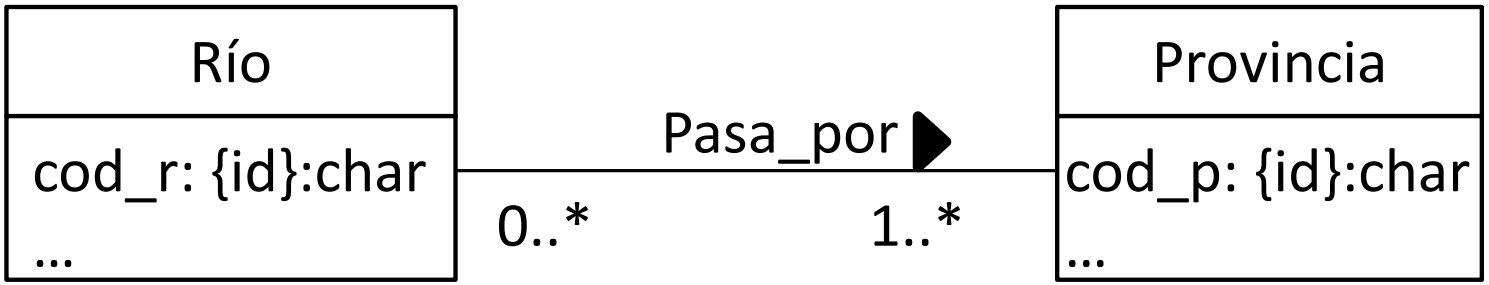
Cardinalidad: Valores más usuales:

- min_A=1 y max_A=1
- min_A=0 y max_A=*
- min_A=0 y max_A=1
- min_A=1 y max_A=*

Cuando la cardinalidad mínima de una clase es distinta de 0 se dice que esa clase tiene restricción de existencia respecto a la asociación.

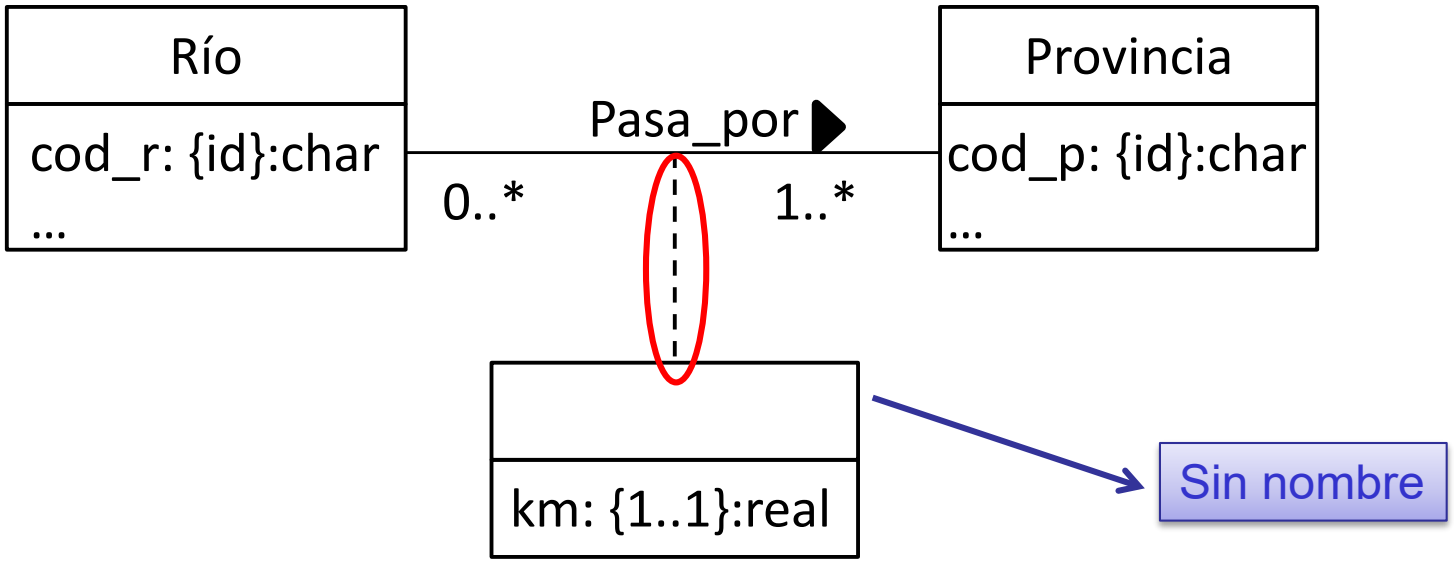
2.1.3.- Asociación (ejemplo)

Diagrama que representa la información de qué ríos pasan por qué provincias. Un río puede pasar por varias provincias, al menos debe pasar por una, y que por una provincia puede que pasen varios ríos o ninguno.



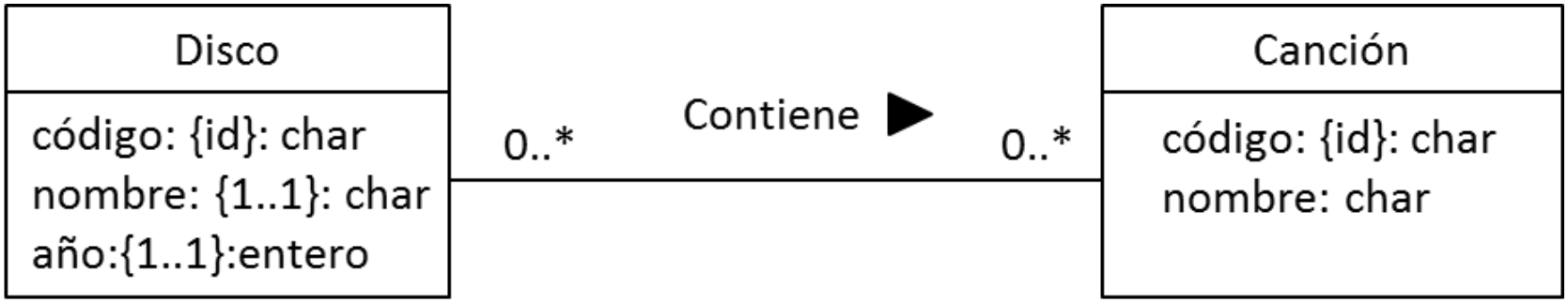
2.1.3.- Asociación como clase (atributo de enlace)

Si se quiere incluir la información: “durante cuántos kilómetros pasa cada río por cada provincia que atraviesa”, la asociación *Pasa_por* debe definirse como una clase anónima porqué tiene un atributo (llamado *de enlace)*



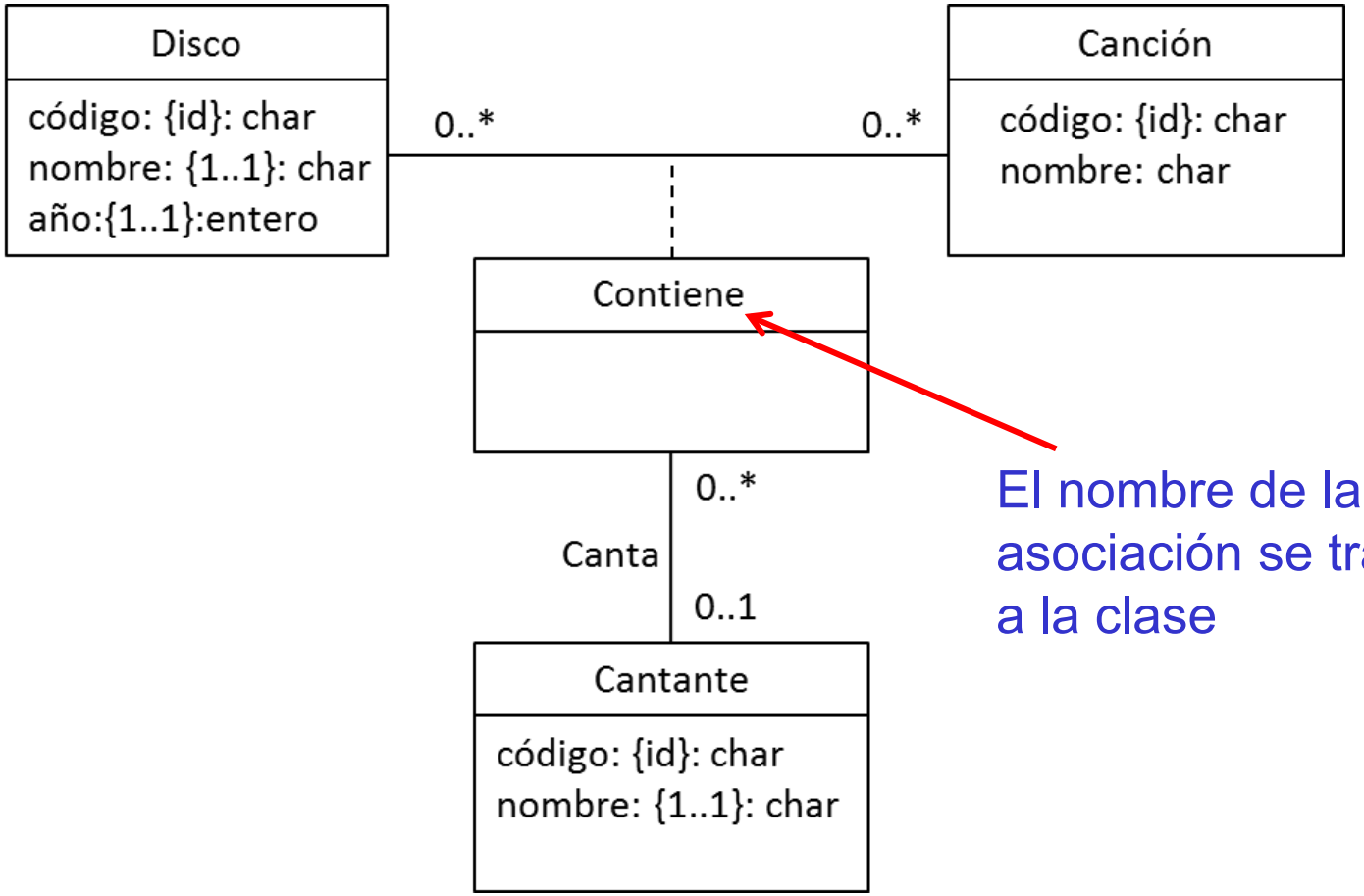
2.1.3.- Asociación como clase (clase asociación)

El diagrama representa la información sobre discos y canciones y las canciones que contiene cada disco.



2.1.3.- Asociación como clase (clase asociación)

…además se quiere información sobre cantantes y "qué cantante canta una determinada canción en un disco"



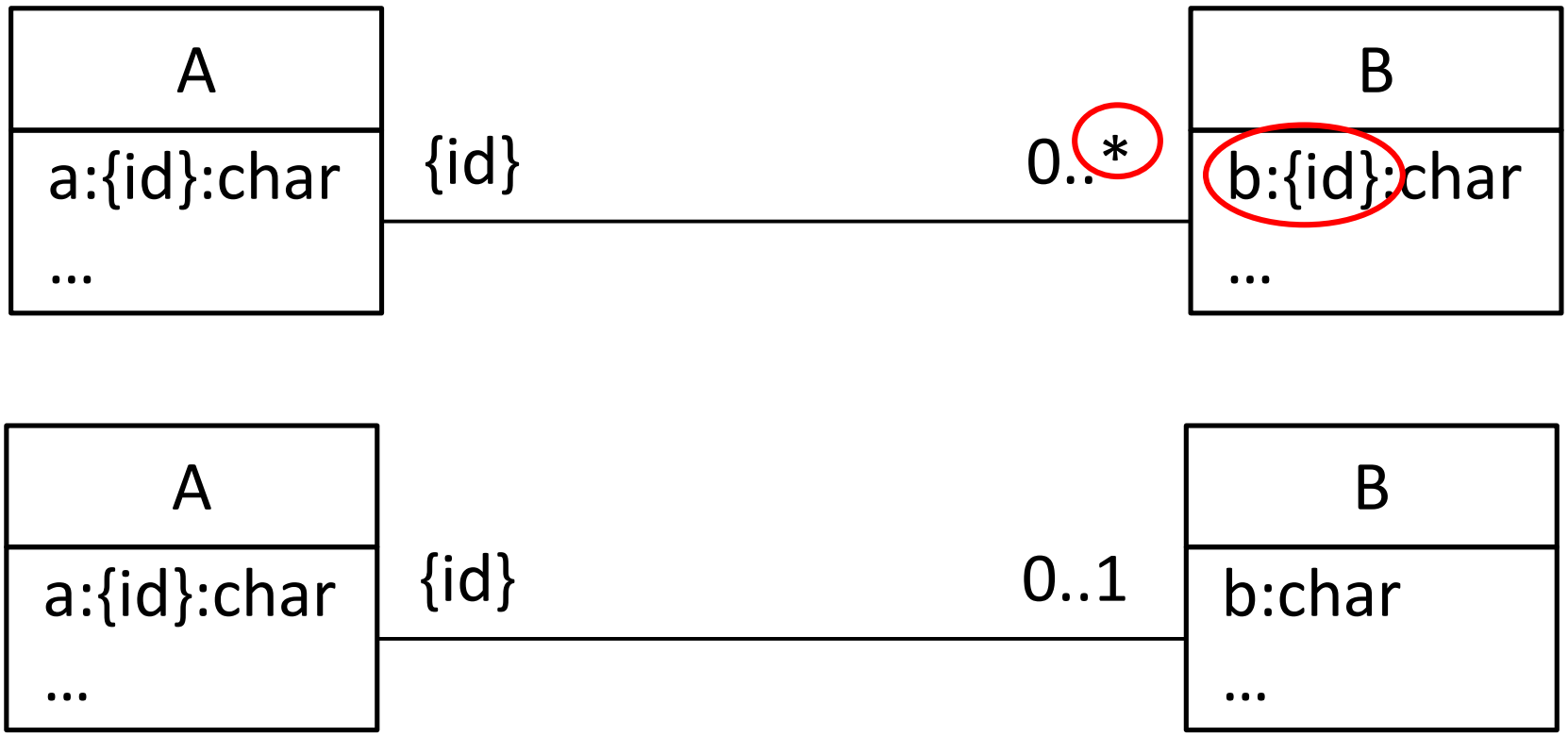
2.1.4.- Clases débiles

Una clase sufre restricción de dependencia de identificación cuando no puede identificarse con sus propios atributos. Sus ocurrencias son distinguibles gracias a su asociación con una o varias clases.

A este tipo de clases se les denomina clases débiles.

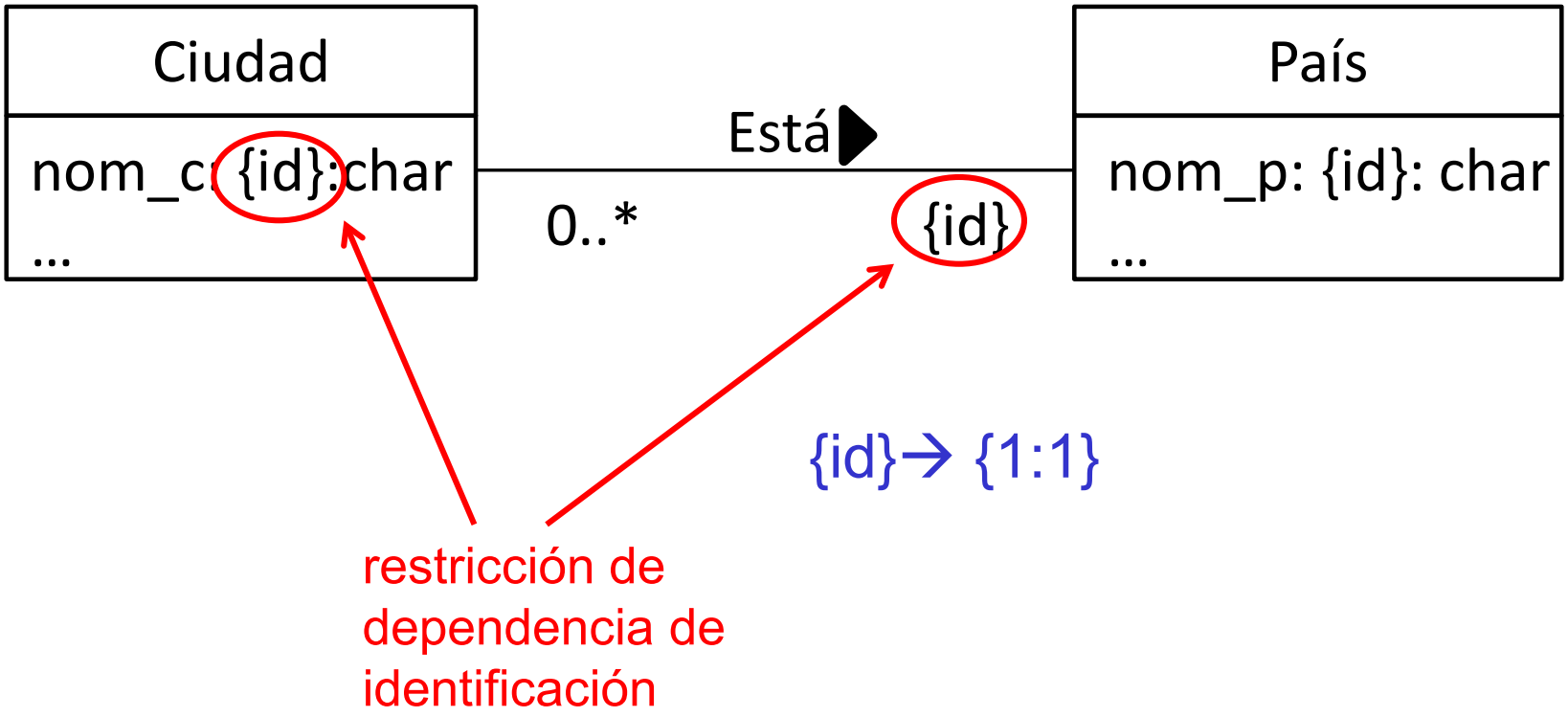
Esta restricción se representa con la etiqueta {id} en lugar de la cardinalidad e implica que la clase débil cuenta entre sus atributos con el o los atributos identificadores de la clase o clases de las que depende, sin que tengan que representarse explícitamente.

2.1.4.- Clases débiles

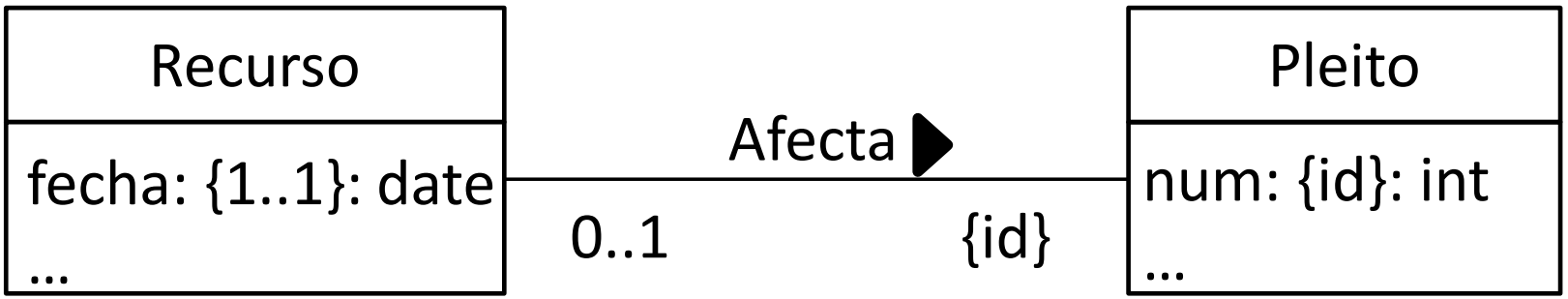


Clases débiles

2.1.4.- Clases débiles (ejemplo)



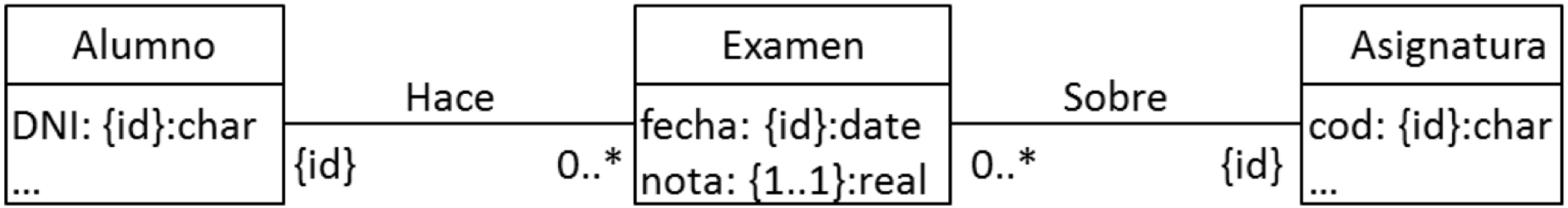
2.1.4.- Clases débiles (ejemplo)



Un pleito solo puede tener un recurso:
Se puede identificar un recursos por el pleito al que pertenece

2.1.4.- Clases débiles (ejemplo)

Se quiere almacenar información sobre las notas que han obtenido los alumnos en distintas asignaturas teniendo en cuenta que un alumno se puede presentar varias veces a una asignatura, pero en fechas distintas, y que los exámenes no tienen identificador.



2.1.5.- Asociaciones ternarias

Aunque en el diagrama de clases de UML podrían representarse explícitamente este tipo de relaciones de grado mayor que dos vamos a utilizar sólo asociaciones binarias con:

- clases asociación o
- clases débiles.

2.1.5.- Asociaciones ternarias (ejemplo)

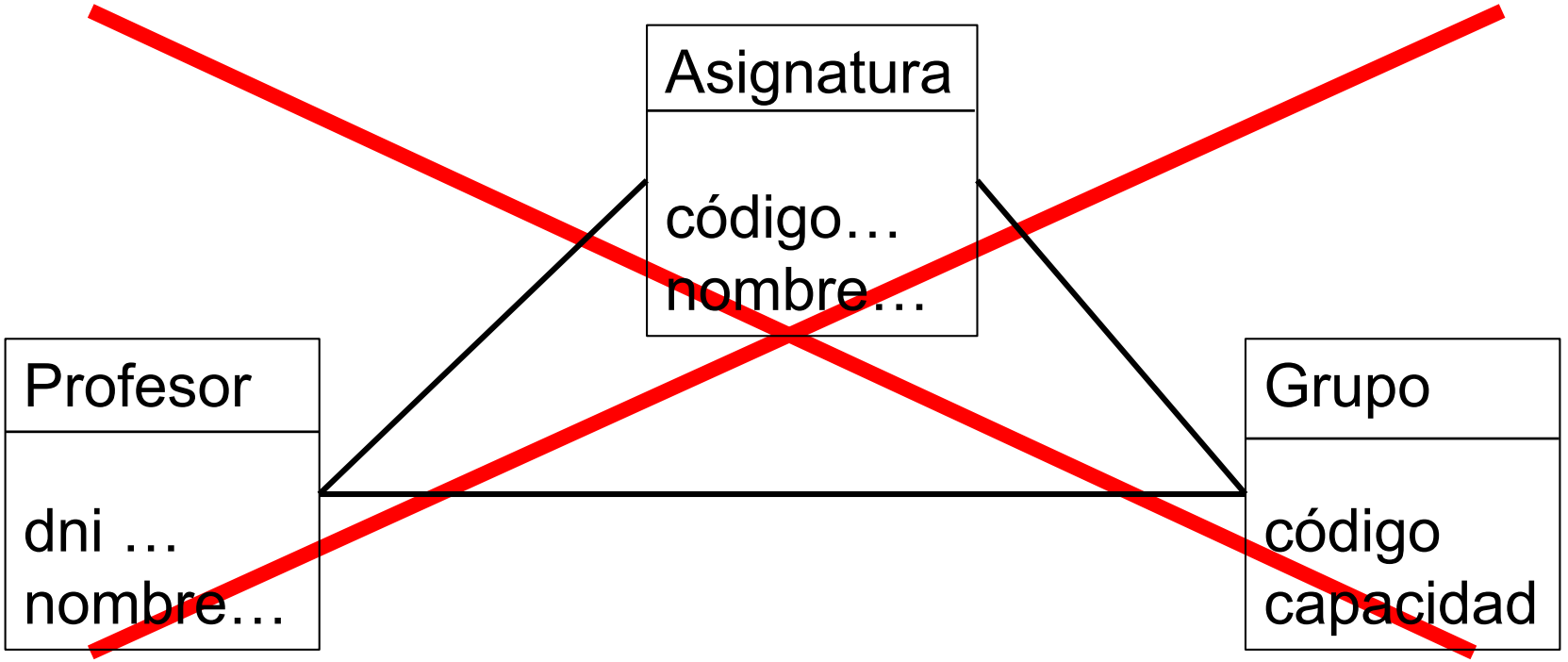
Ejemplo:

Asociación ternaria entre profesor-asignatura-grupo:

- Un profesor puede impartir cualquier número de asignaturas en cualquier número de grupos
- Una asignatura puede ser impartida por cualquier número de profesores y de grupos
- Un grupo puede tener cualquier número de asignaturas impartidas por cualquier número de profesores

2.1.5.- Asociaciones ternarias (ejemplo)

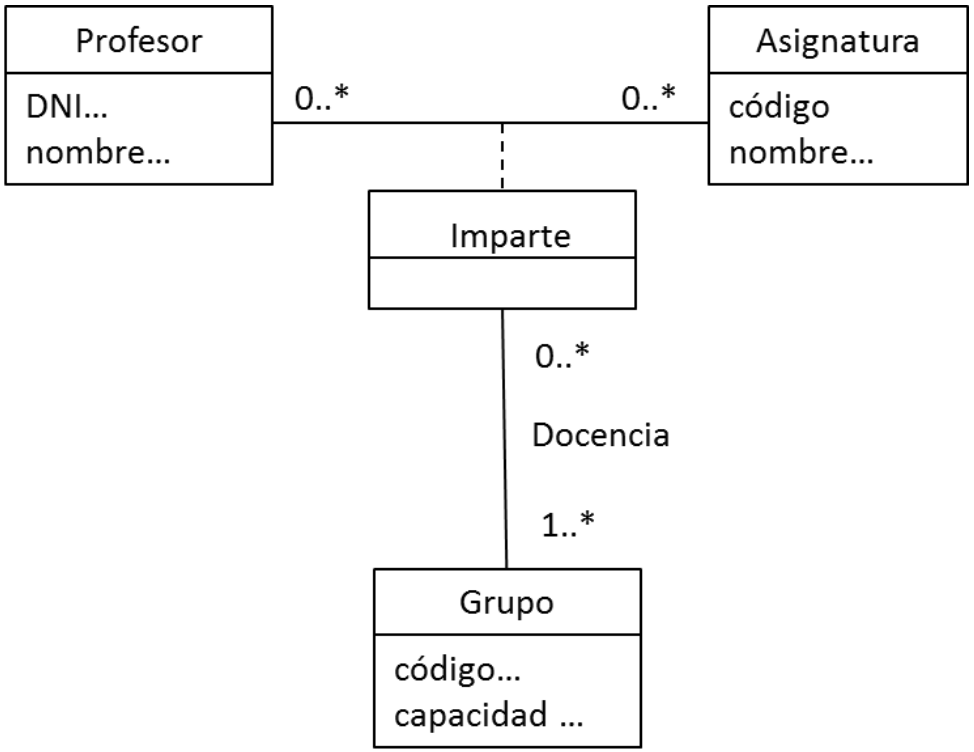
- No se puede modelar como 3 asociaciones binarias porque perderíamos información.



Un profesor imparte varias asignaturas. Un profesor imparte varios grupos pero ¿en qué grupo imparte cada asignatura un profesor ?

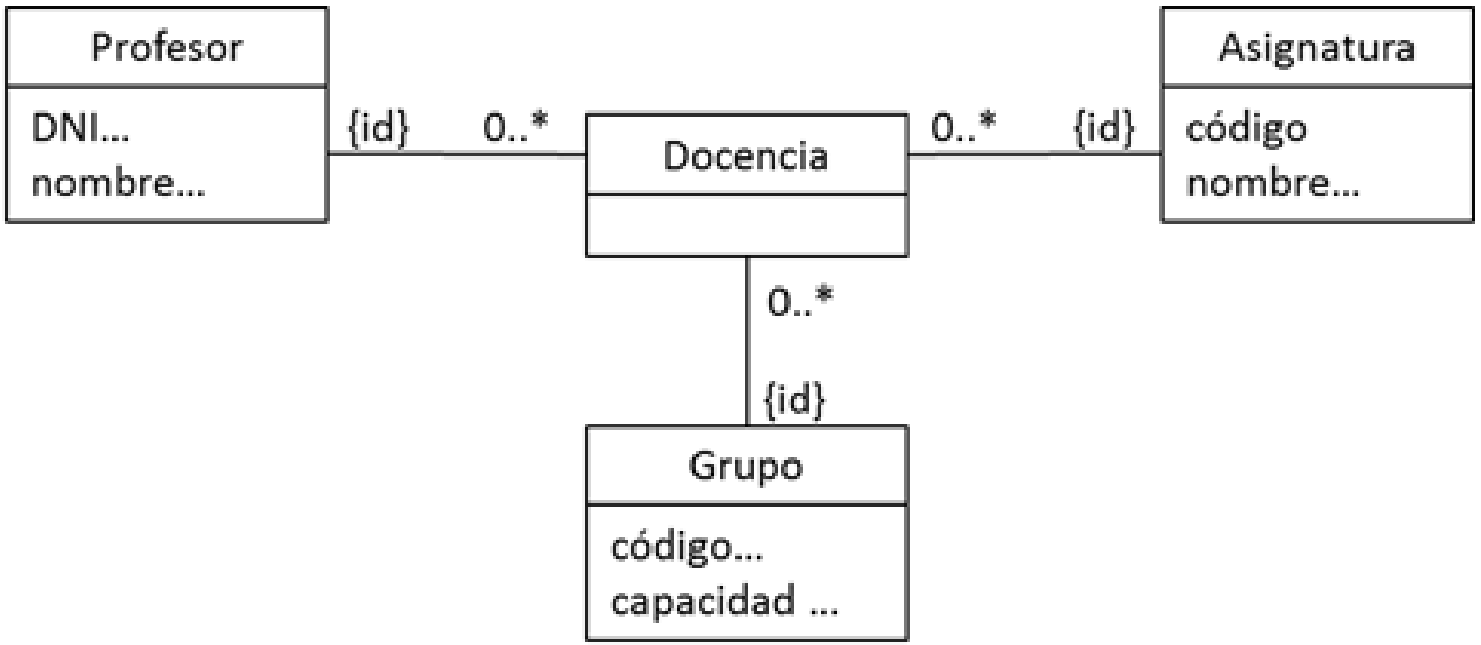
2.1.5.- Asociaciones ternarias (ejemplo)

Asociación ternaria entre profesor-asignatura-grupo representada usando una clase asociación *Imparte*



2.1.5.- Asociaciones ternarias (ejemplo)

Asociación ternaria entre profesor-asignatura-grupo representada usando una clase débil *Docencia*



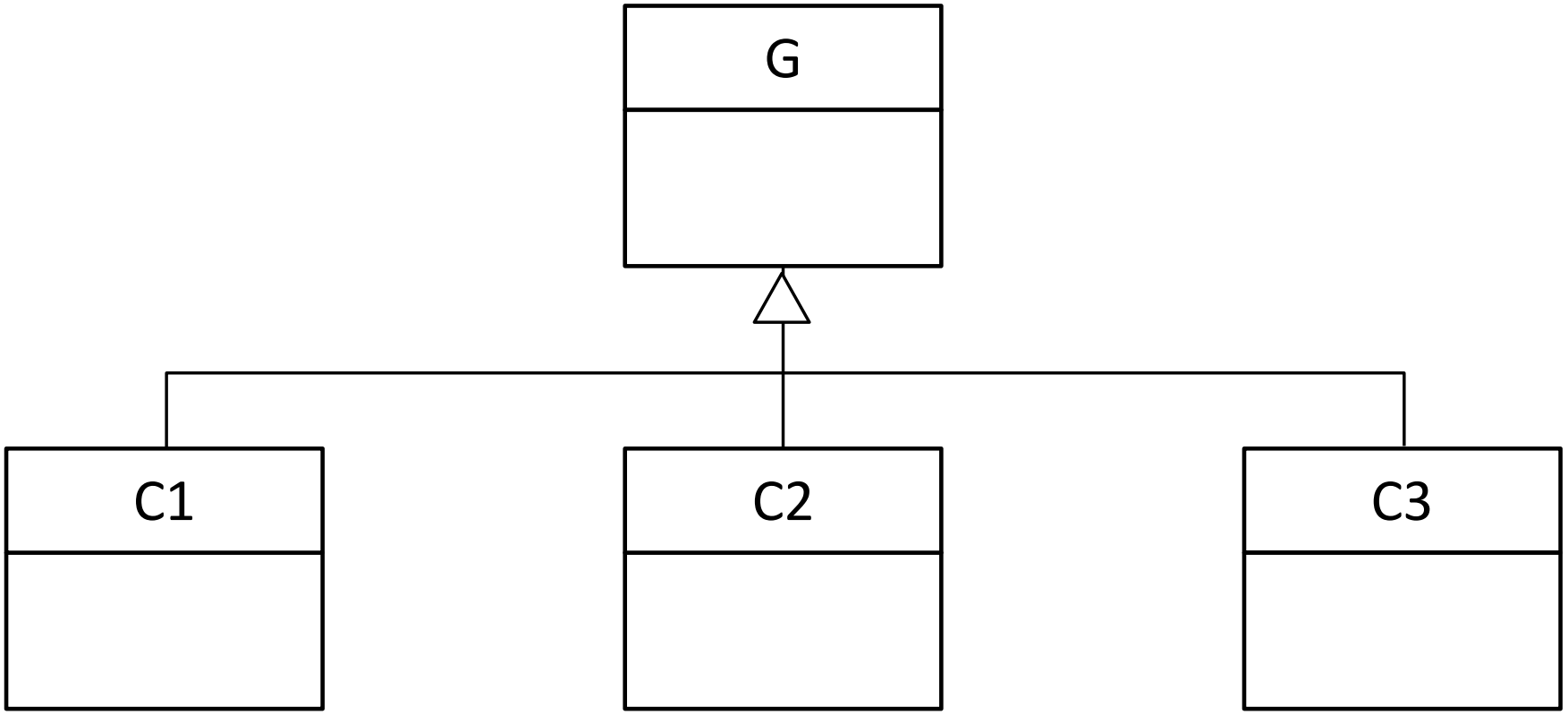
2.1.6.- Generalización/Especialización

Cuando se detecta que entre distintas clases definidas en el esquema existe una relación de inclusión este hecho se expresa por medio de la Generalización/Especialización.

La clase más general se especializa en una o varias subclases, o dicho a la inversa, que una o varias clases se generalizan en una superclase.

Las subclases tienen, además de sus atributos, todos los atributos de sus superclases (en cualquier nivel), aunque no se representan en el diagrama.

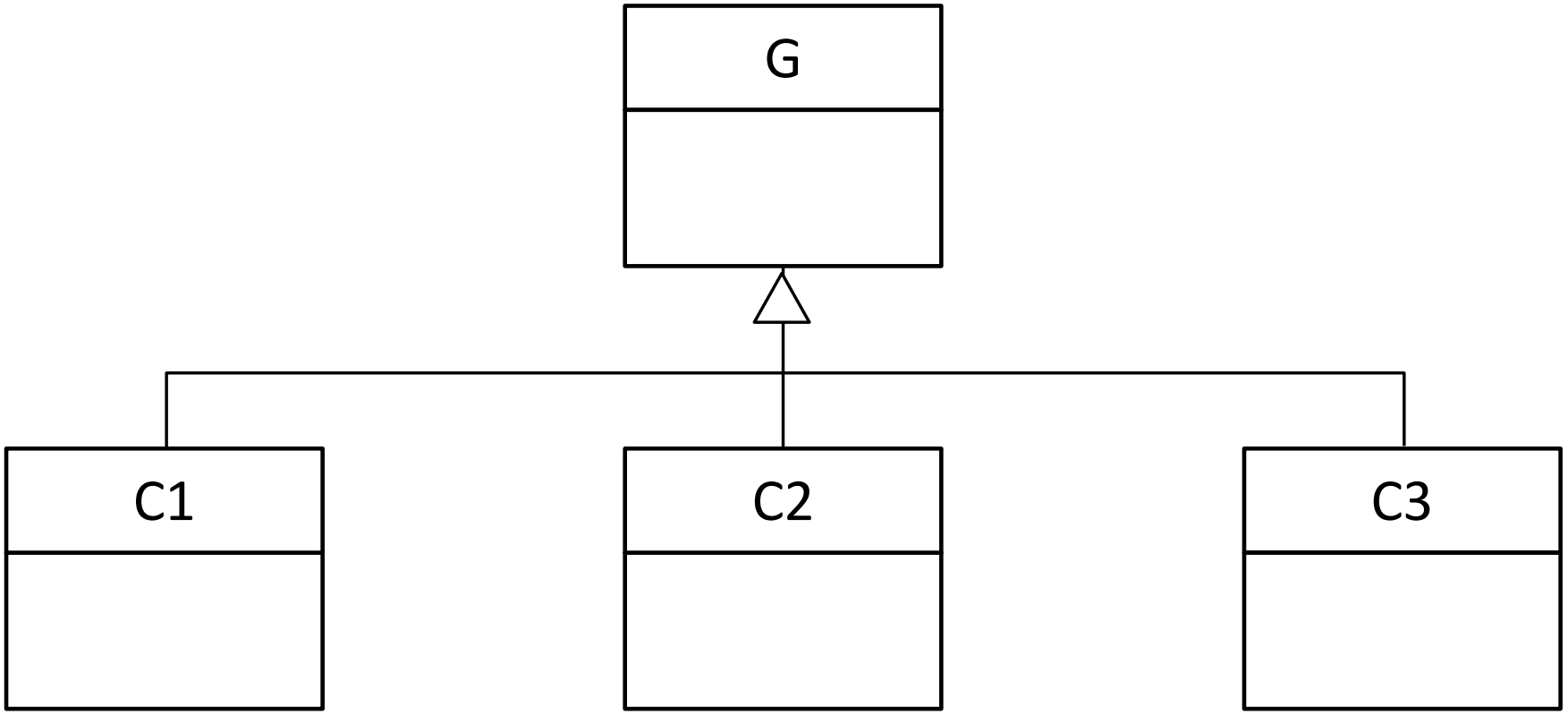
2.1.6.- Generalización/Especialización



Total: toda ocurrencia de la clase *G* se especializa en *C1*, *C2* o *C3*

Parcial: puede haber ocurrencias de *G* que no se especialicen ni en *C1* ni en *C2* ni en *C3* *(Valor por defecto)*

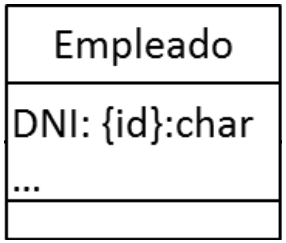
2.1.6.- Generalización/Especialización



Disjunta: no puede haber ocurrencias de *G* que estén a la vez en dos subclases

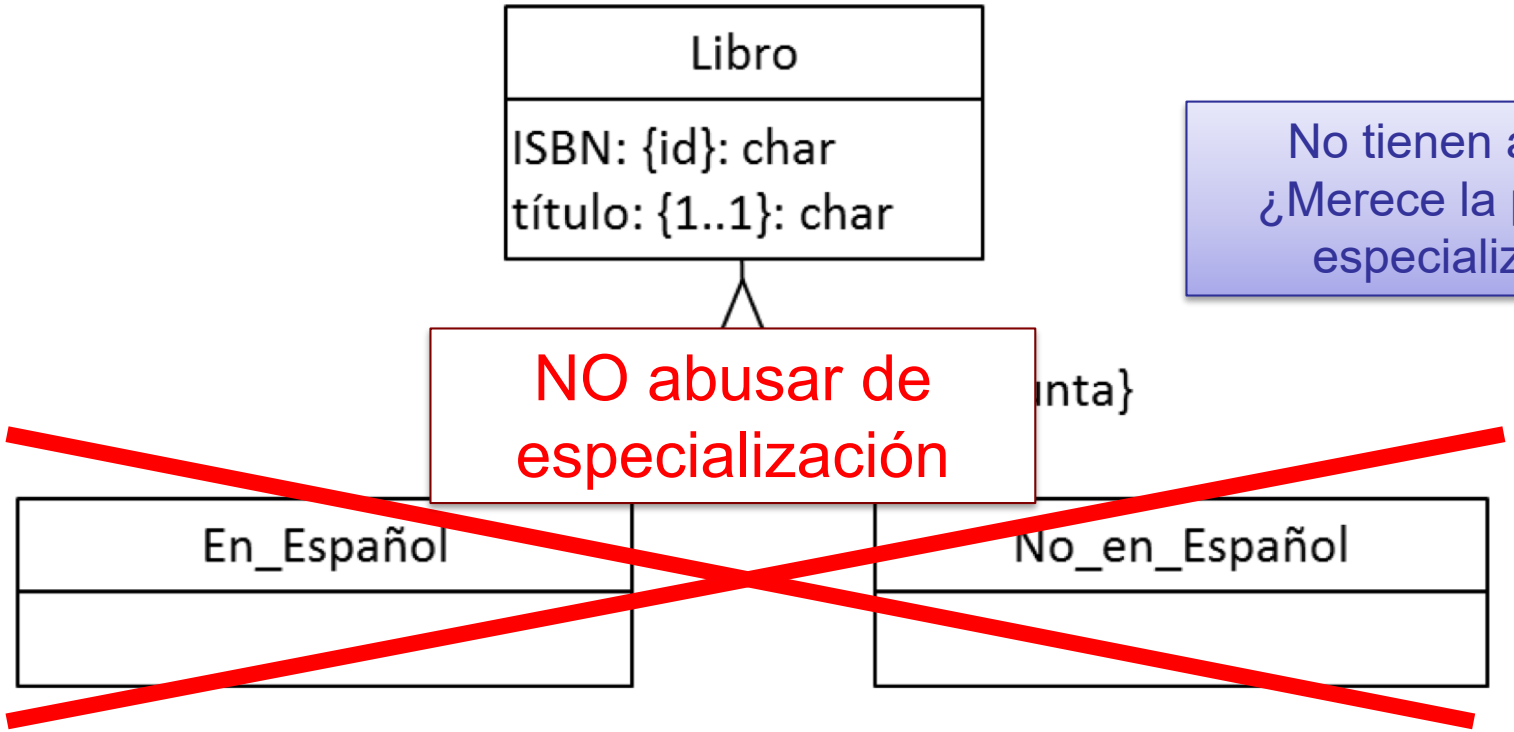
Solapada: puede haber ocurrencias de *G* que se especialicen en más de una subclase *(Valor por defecto)*

2.1.6.- Generalización/Especialización



2.1.6.- Generalización/Especialización

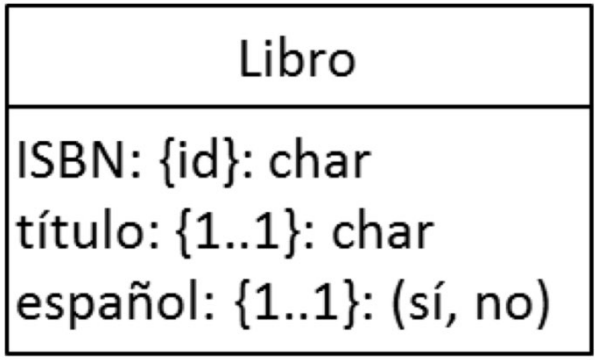
En una biblioteca se almacena de cada libro su ISBN y su título.
Los libros pueden estar escritos en español o en cualquier otra lengua.



No tienen atributos
¿Merece la pena esta especialización?

2.1.6.- Generalización/Especialización

En una biblioteca se almacena de cada libro su ISBN y su título.
Los libros pueden estar escritos en español o en cualquier otra lengua.



UD 4.2.- Diseño Conceptual

2.2.- Obtención del diagrama de clases

- Identificar las clases con sus atributos,
- Identificar generalizaciones/especializaciones,
- Identificar asociaciones entre clases,
- Especificar restricciones de integridad, y
- Comprobaciones finales.

2.2.1.- Identificar clases con sus atributos

Por cada tipo de objeto de la realidad se definirá una clase en el diagrama.

Una clase viene definida por los atributos que representan la información que se desea conocer de cada tipo de objeto.

Para cada atributo se debe:

- Asociar un dominio o, si es derivado, especificar la forma de derivación; y
- Especificar la cardinalidad y las restricciones de unicidad.
- Elegir identificador; si no existe identificador, la clase debe ser considerada débil y habrá que decidir, sobre qué asociaciones se apoya para identificarse.
- Posibilidad de reconsiderar clases elegidas …

2.2.2.- Identificar generalizaciones/especializaciones

La especialización es el proceso por el que se clasifica una clase de objetos en subclases más especializadas.

La generalización es el proceso inverso por el que se generalizan varias clases para obtener una abstracta de más alto nivel que incluya los objetos de todas estas clases.

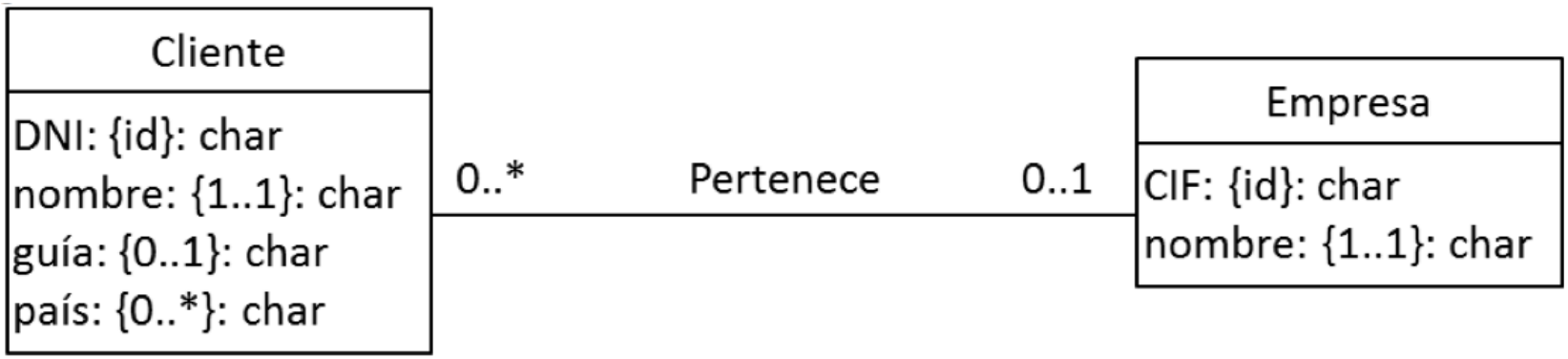
- Estrategia descendente
- Estrategia ascendente
- Jerarquización (es_un)

No abusar de las especializaciones.

2.2.2.- Identificar generalizaciones/especializaciones

Estrategia descendente (especialización)

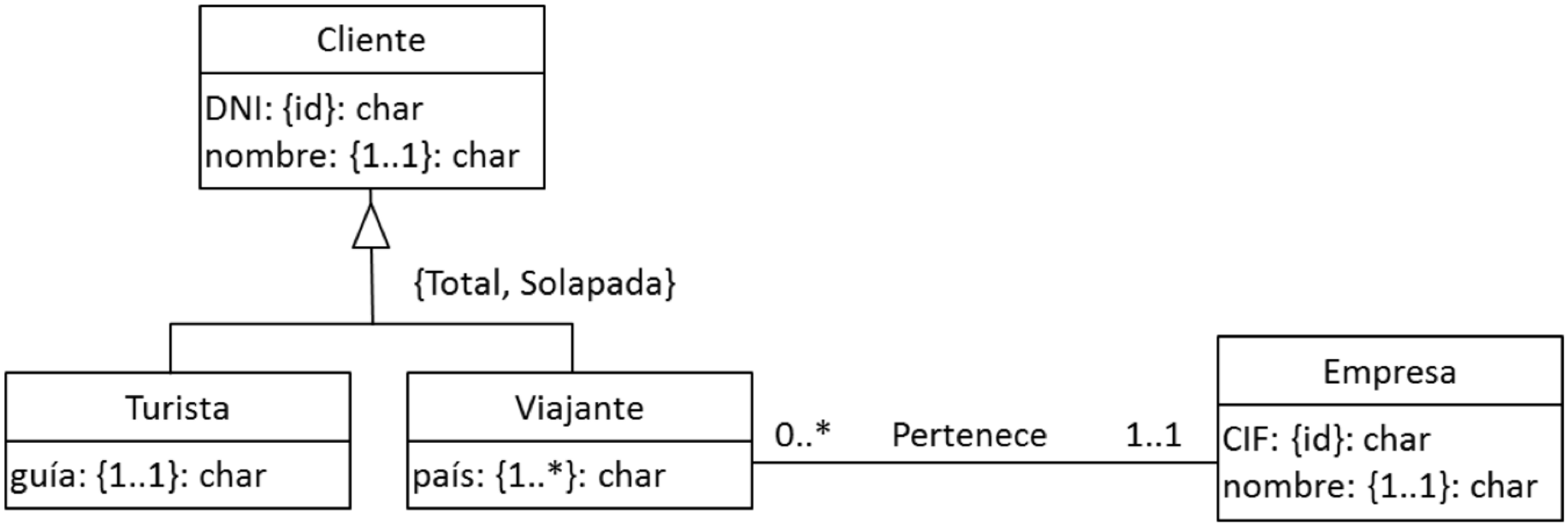
Especialización: proceso por el que se clasifica una clase de objetos en subclases más especializadas.



2.2.2.- Identificar generalizaciones/especializaciones

Estrategia descendente (especialización)

Especialización: proceso por el que se clasifica una clase de objetos en subclases más especializadas.



2.2.2.- Identificar generalizaciones/especializaciones

Estrategia ascendente (generalización)

Generalización: Proceso por el que se generalizan varias clases para obtener una abstracta de más alto nivel que incluya los objetos de todas estas clases.

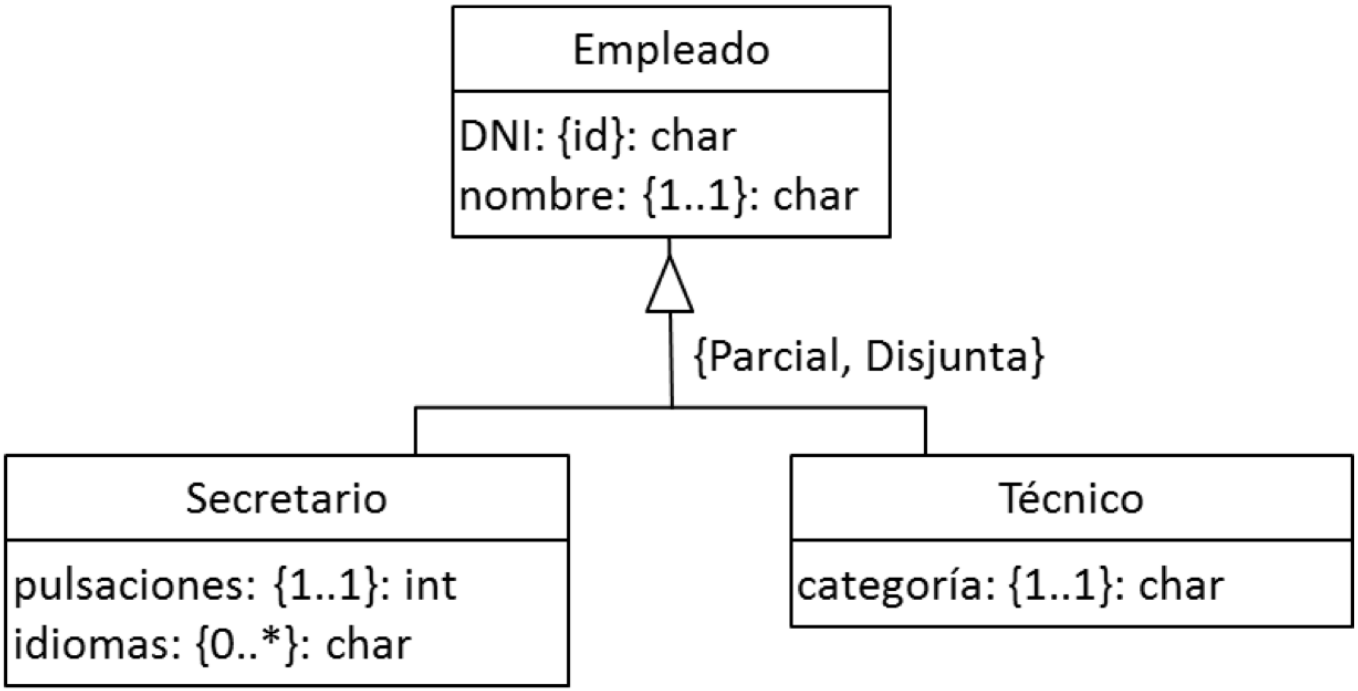
Secretario
DNI: {id}: char nombre: {1..1}: char pulsaciones: {1..1}: int idiomas: {0..*}: char

Técnico
DNI: {id}: char nombre: {1..1}: char categoría: {1..1}: char

2.2.2.- Identificar generalizaciones/especializaciones

Estrategia ascendente (generalización)

Generalización: Proceso por el que se generalizan varias clases para obtener una abstracta de más alto nivel que incluya los objetos de todas estas clases.



2.2.2.- Identificar generalizaciones/especializaciones

Jerarquización (es_un)

Jerarquización: Clases entre las que existe una relación “es_un”

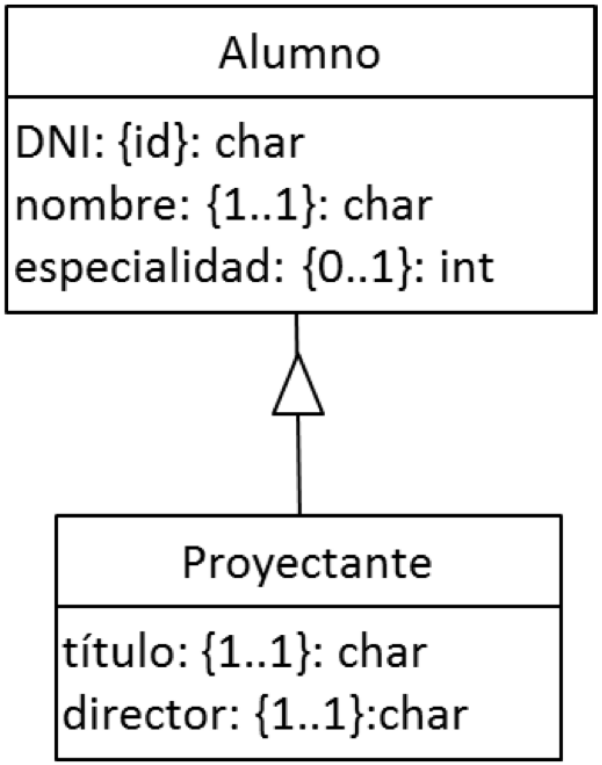
Alumno
DNI: {id}: char nombre: {1..1}: char especialidad: {0..1}: int

Proyectante
DNI: {id}: char nombre: {1..1}: char especialidad: {0..1}: char título: {1..1}: char director: {1..1}:char

2.2.2.- Identificar generalizaciones/especializaciones

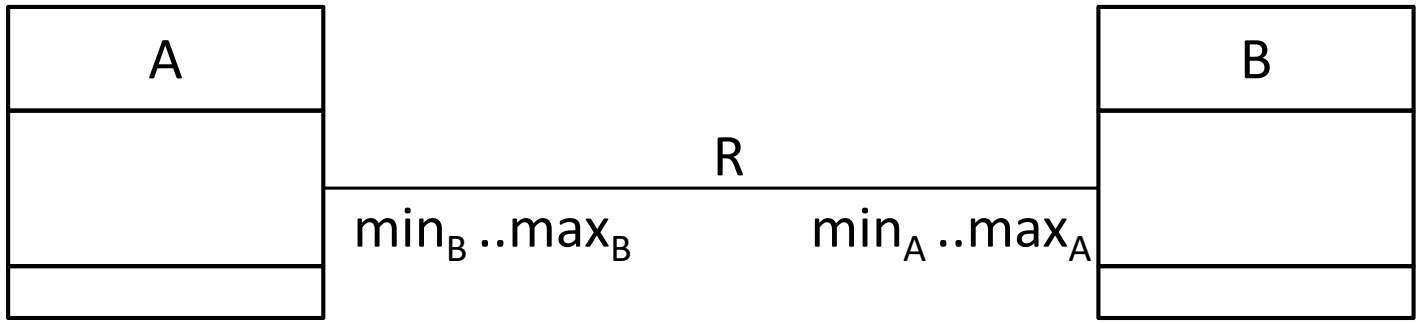
Jerarquización (es_un)

Jerarquización: Clases entre las que existe una relación “es_un”



2.2.3.- Identificar asociaciones entre clases

Una vez definido un conjunto inicial de clases hay que estudiar las asociaciones existentes entre ellas y las cardinalidades máximas y mínimas.



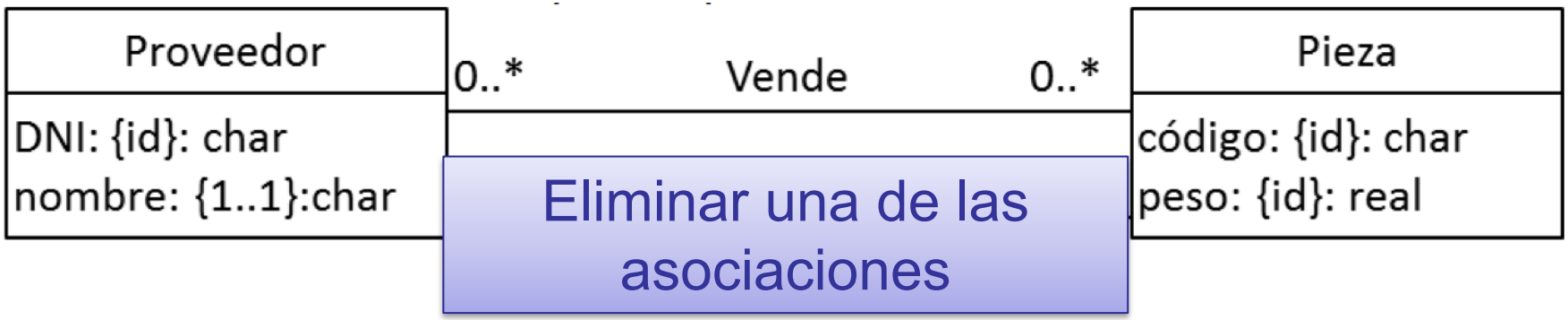
- Dado un elemento de la clase A, ¿con cuántos elementos de la clase B debe asociarse como mínimo? = min_A
- Dado un elemento de la clase A, ¿con cuántos elementos de la clase B puede asociarse como máximo? = max_A
- Dado un elemento de la clase B, ¿con cuántos elementos de la clase A debe asociarse como mínimo? = min_B
- Dado un elemento de la clase B, ¿con cuántos elementos de la clase A puede asociarse como máximo? = max_B

2.2.3.- Identificar asociaciones entre clases

1. Ante la duda, elegir las cardinalidades menos restrictivas
2. Eliminar asociaciones redundantes
3. Eliminar redundancias por ciclos
4. Especificar roles en asociaciones de una clase consigo misma

2.2.3.- Identificar asociaciones entre clases

Eliminar asociaciones redundantes



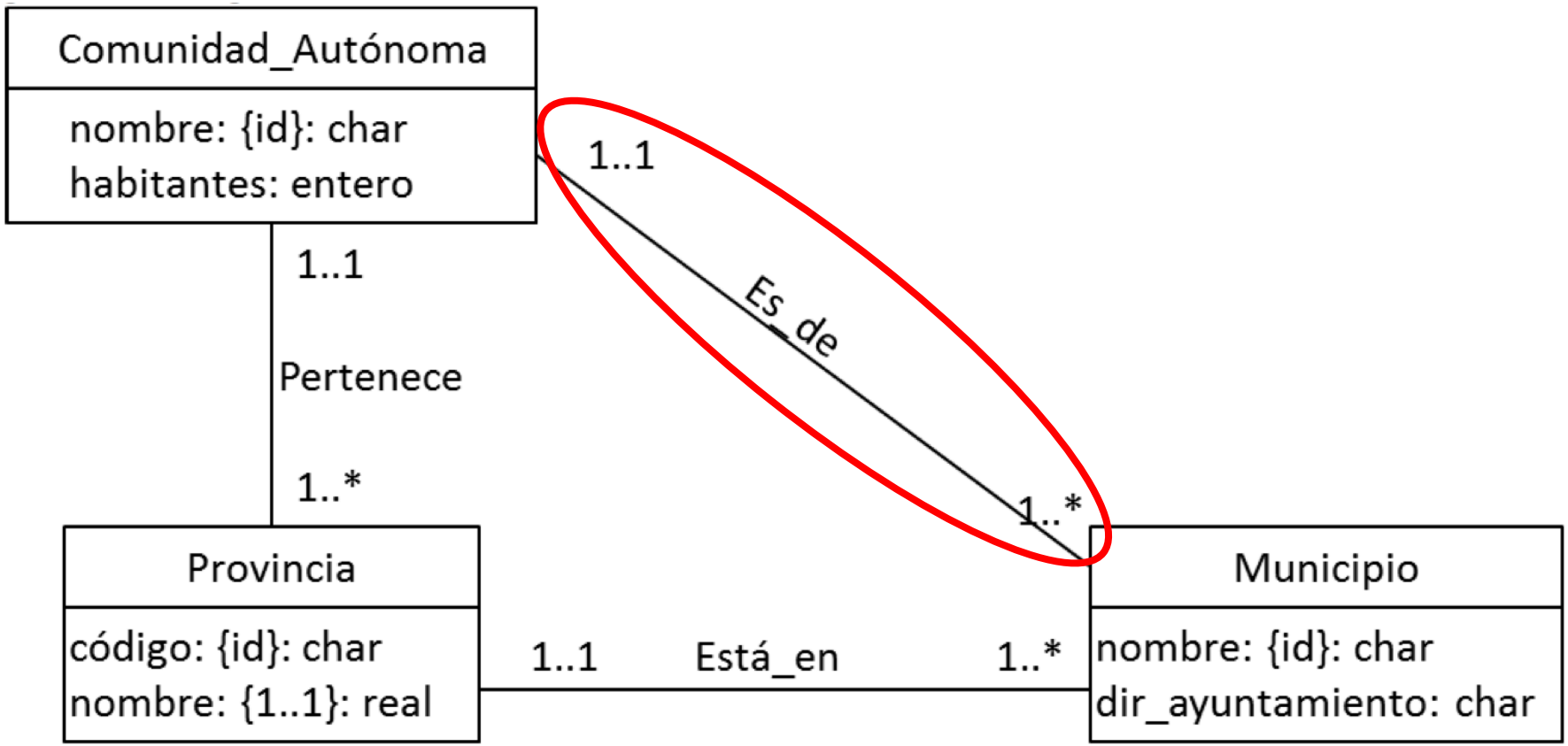
2.2.3.- Identificar asociaciones entre clases

Eliminar asociaciones redundantes

Las 2 asociaciones son necesarias:
Representan información distinta

2.2.3.- Identificar asociaciones entre clases

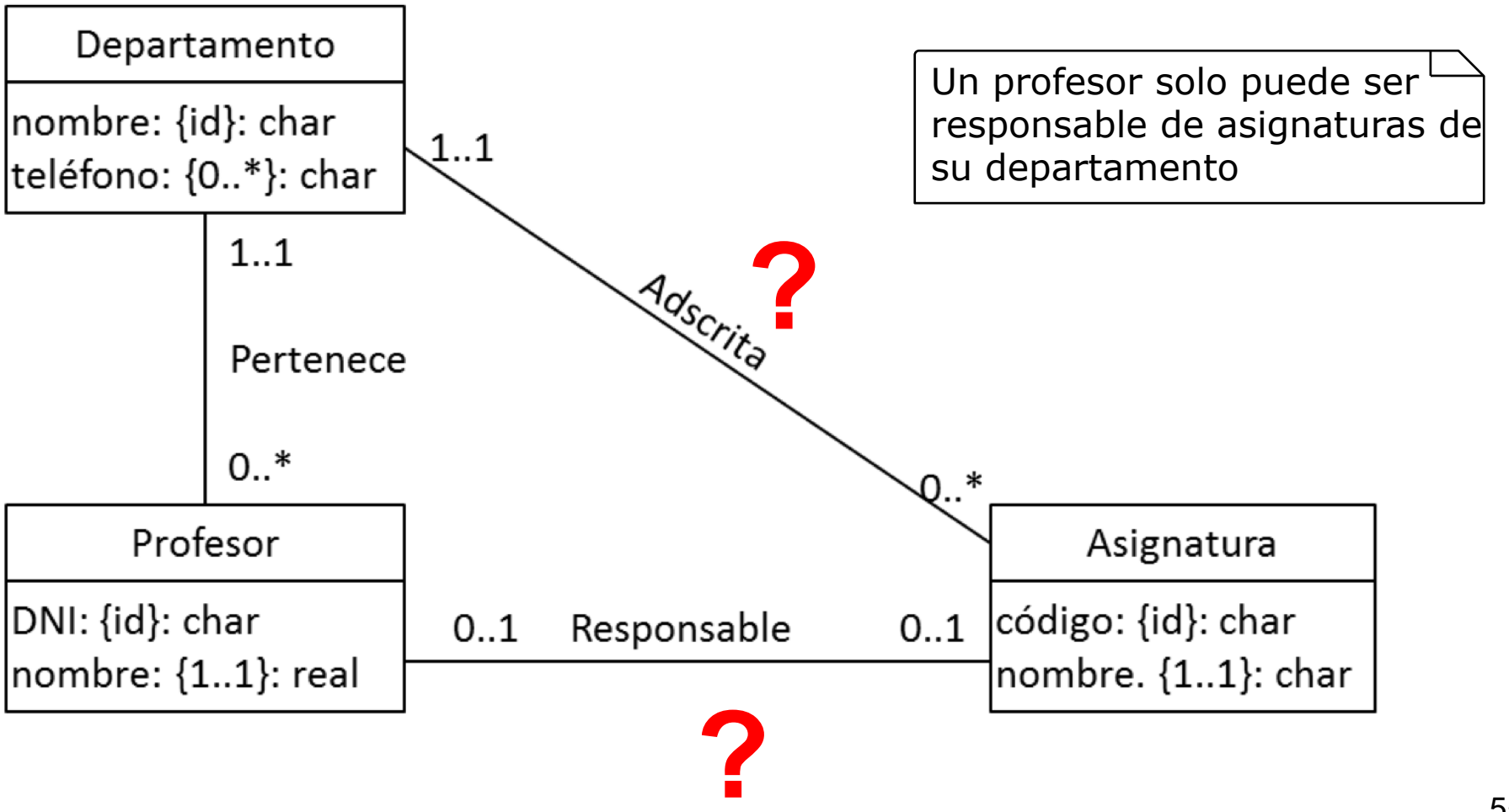
Eliminar redundancias por ciclos



2.2.3.- Identificar asociaciones entre clases

Eliminar redundancias por ciclos

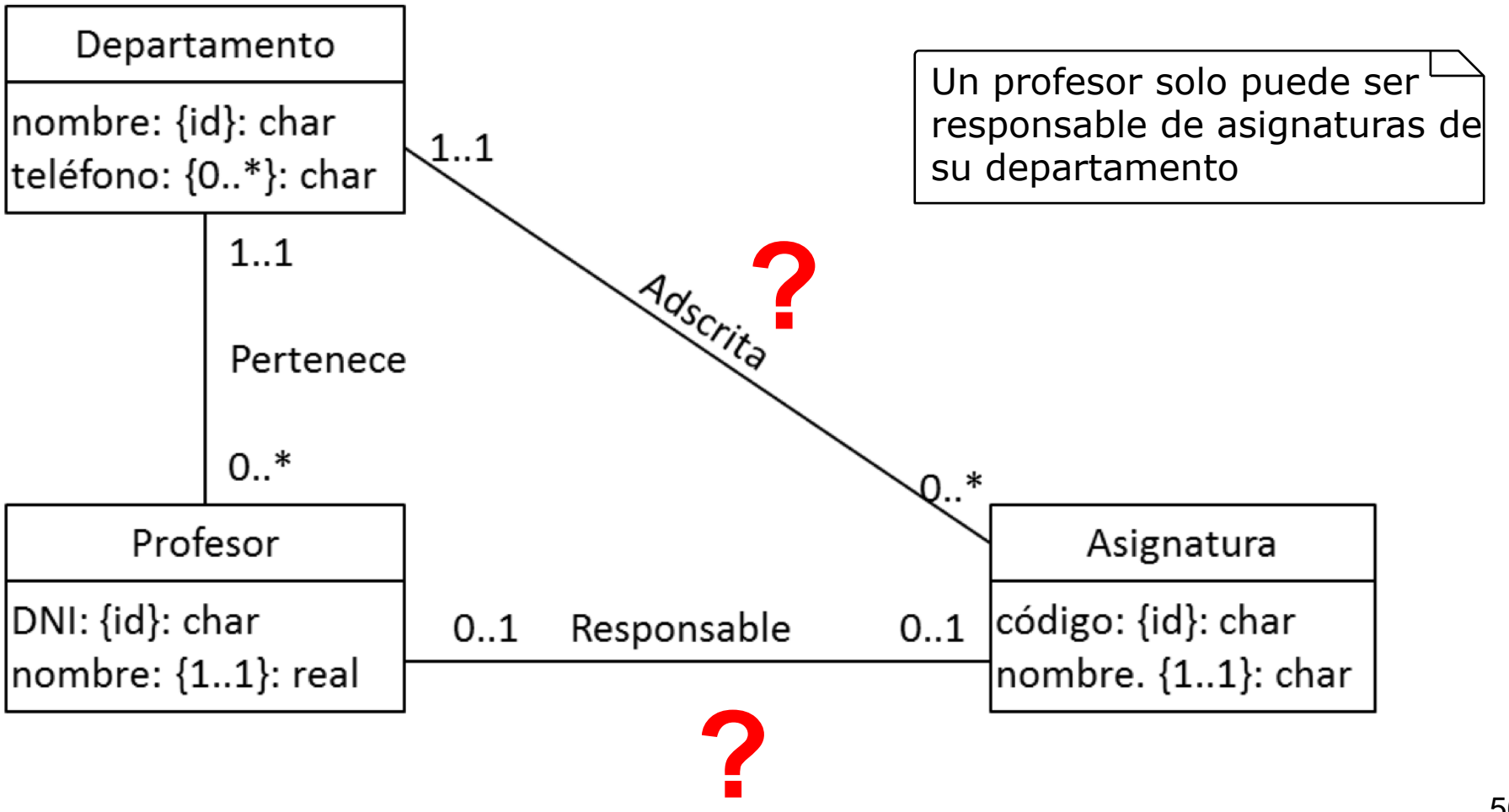
No siempre puede eliminarse



2.2.3.- Identificar asociaciones entre clases

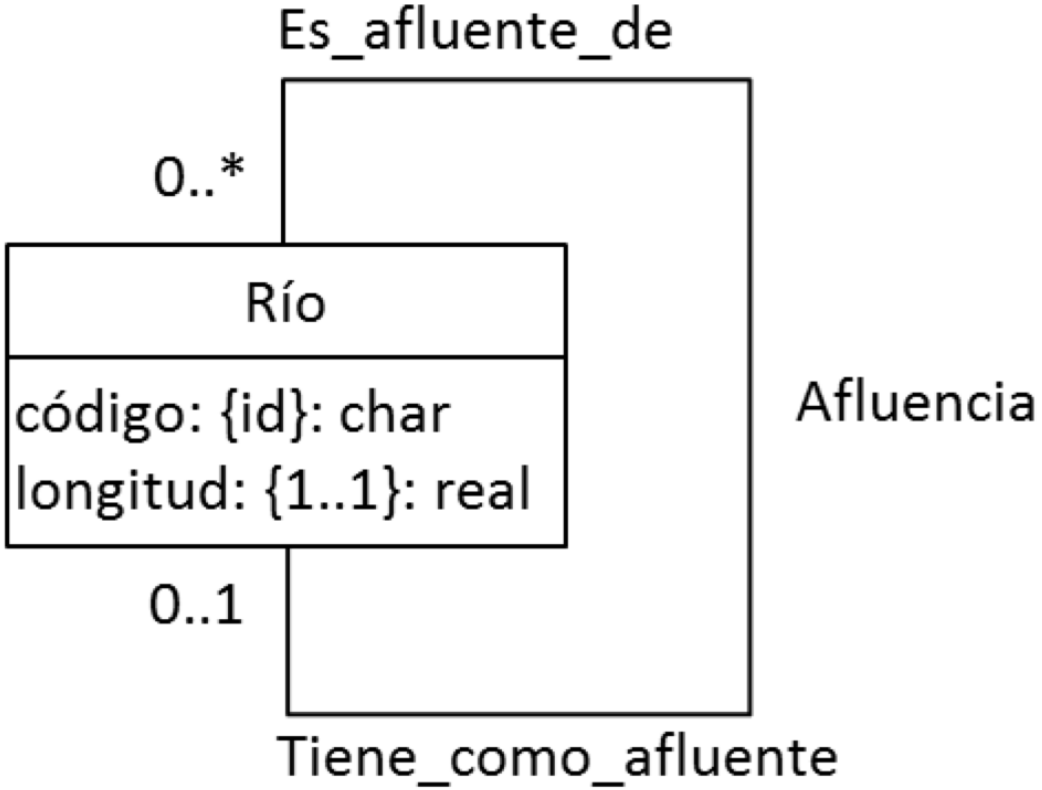
Eliminar redundancias por ciclos

No siempre puede eliminarse



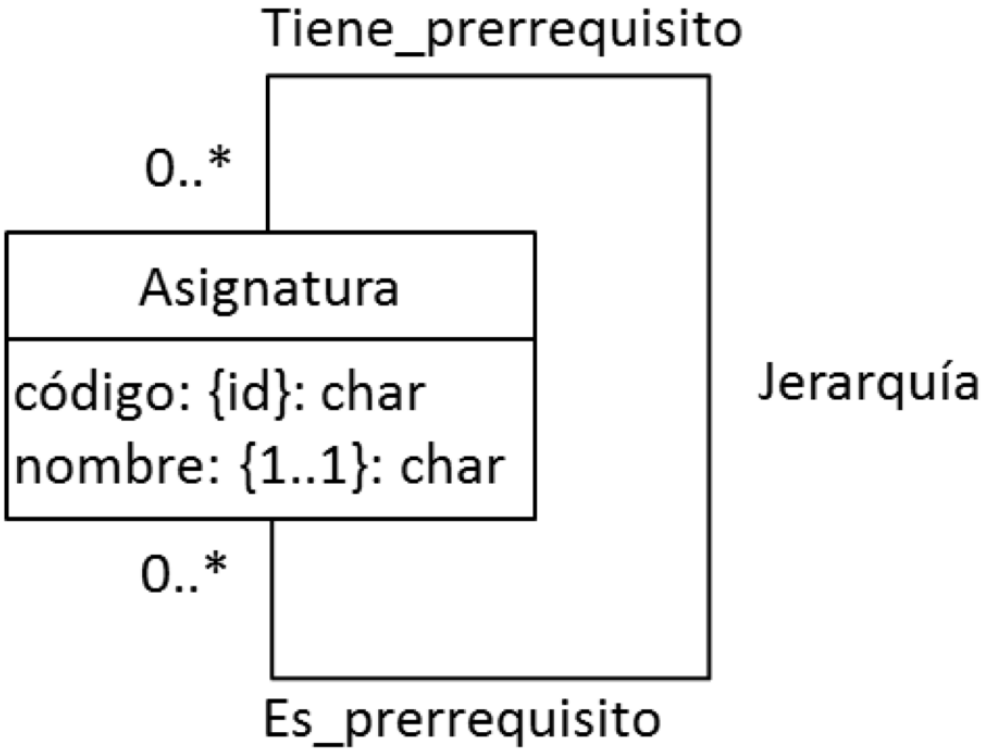
2.2.3.- Identificar asociaciones entre clases

Especificar roles en asociaciones de una clase consigo misma



2.2.3.- Identificar asociaciones entre clases

Especificar roles en asociaciones de una clase consigo misma



2.2.3.- Identificar asociaciones entre clases

Especificar roles en asociaciones de una clase consigo misma

Cuidado con las asociaciones reflexivas:
Normalmente exigen que se especifiquen ciertas propiedades que no quedan contempladas en la definición de la asociación.

Ejemplo anterior (prerrequisitos)

- Es antisimétrica:
  Una asignatura no puede ser prerrequisito de un prerrequisito suyo

- Es antirreflexiva:
  Una asignatura no puede ser prerrequisito de sí misma

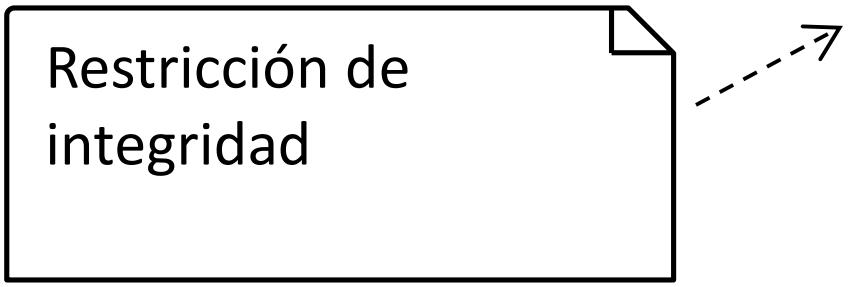
Estas propiedades no están expresadas en el diagrama.

2.2.4.- Especificar restricciones de integridad

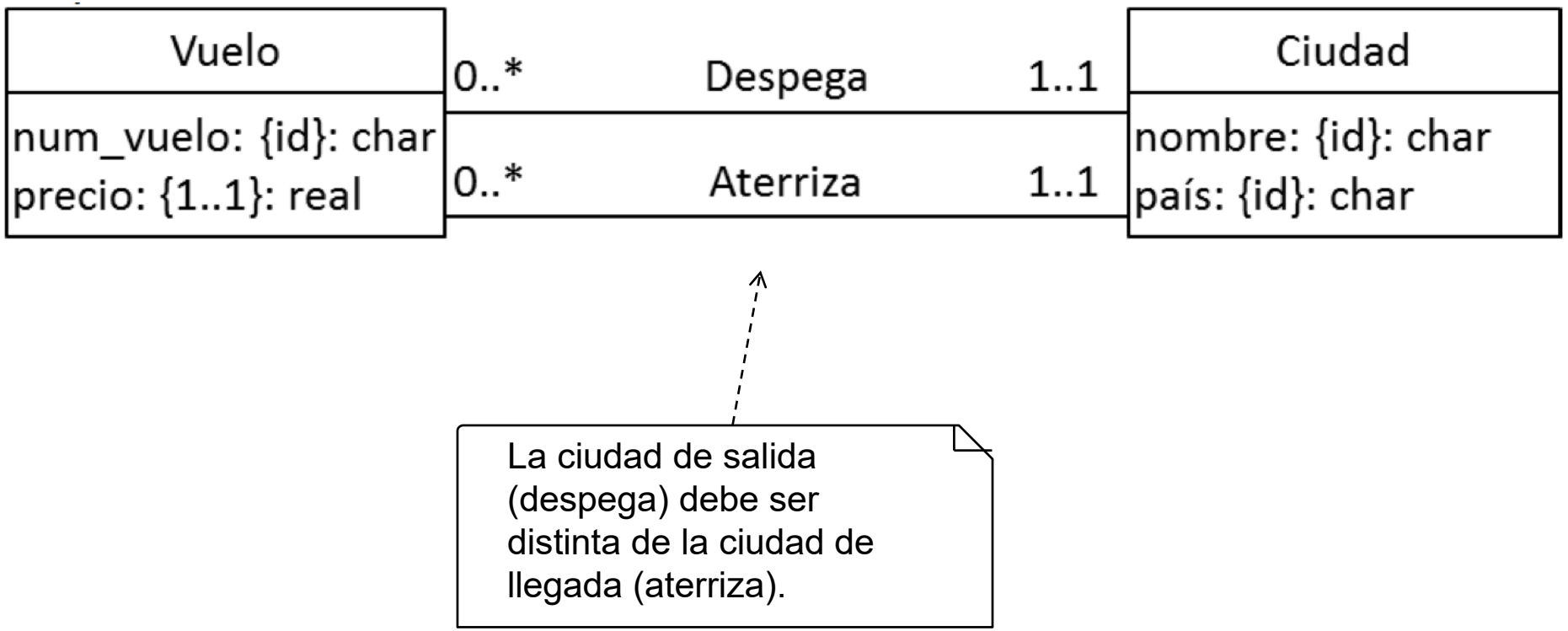
Todas aquellas propiedades de la realidad que no hayan quedado expresadas en el diagrama de clases deben incluirse.

Utilizar los elementos de anotación de los diagramas de UML.

Los elementos de anotación son comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento del modelo.



2.2.4.- Especificar restricciones de integridad



2.2.5.-Comprobaciones finales

- Nombres:
  - No se usa el mismo nombre para distintas clases, asociaciones o roles
  - No se repiten nombres de atributos en una clase/subclase

- Identificadores
  - Toda clase tiene identificador (o es débil o es subclase)
  - Las clases asociación no tienen identificadores
  - Una clase no tiene atributos para referirse a otras clases: Se usan asociaciones

UD 4.2.- Diseño Conceptual

2.3.- Ejercicio 1: Restaurante

"Un restaurante desea que sus menús puedan ser consultados desde terminales y para ello ha decidido diseñar una base de datos relacional. En el restaurante se ofertan varios menús de los que se quiere saber el código, el precio y los platos que lo componen (al menos uno); cada plato puede haber sido diseñado como mucho por un cocinero del restaurante (que tiene DNI, nombre, país de origen y edad como datos de interés); de los platos se desea saber el código asignado al plato, el nombre y las calorías aproximadas; también puede interesar almacenar para algunos platos los ingredientes necesarios (indicando la cantidad) y el vino que se recomienda para su completo disfrute. De cada vino disponible en el restaurante se quiere conocer su código (interno al restaurante), el tipo, el nombre y la añada. Por último y para poder controlar la despensa, de cada ingrediente es interesante almacenar un código, el nombre, el precio en mercado y las existencias que quedan (obligatorio)."

2.3.- Ejercicio 1: Restaurante

1. Identificar las clases con sus atributos

"Un restaurante desea que sus menús puedan ser consultados desde terminales y para ello ha decidido diseñar una base de datos relacional. En el restaurante se ofertan varios menús de los que se quiere saber el código, el precio y los platos que lo componen (al menos uno); cada plato puede haber sido diseñado como mucho por un cocinero del restaurante (que tiene DNI, nombre, país de origen y edad como datos de interés); de los platos se desea saber el código asignado al plato, el nombre y las calorías aproximadas; también puede interesar almacenar para algunos platos los ingredientes necesarios (indicando la cantidad) y el vino que se recomienda para su completo disfrute. De cada vino disponible en el restaurante se quiere conocer su código (interno al restaurante), el tipo, el nombre y la añada. Por último y para poder controlar la despensa, de cada ingrediente es interesante almacenar un código, el nombre, el precio en mercado y las existencias que quedan (obligatorio)."

2.3.- Ejercicio 1: Restaurante

1. Identificar las clases con sus atributos

Menú
código: {id}: char precio: real

Cocinero
DNI: {id}: char nombre: {1..1}: char país: char edad: entero

Plato
código: {id}: char nombre :{1..1} : char calorías: entero

Vino
código: {id}: char nombre :{1..1}:: char añada: año tipo: {tinto, blanco, rosado}

Ingrediente
código: {id}: char nombre :{1..1}: char Precio{1..1}: real existencias {1..1}: real

2.3.- Ejercicio 1: Restaurante

2. Identificar generalizaciones/especializaciones

"Un restaurante desea que sus menús puedan ser consultados desde terminales y para ello ha decidido diseñar una base de datos relacional. En el restaurante se ofertan varios menús de los que se quiere saber el código, el precio y los platos que lo componen (al menos uno); cada plato puede haber sido diseñado con un cocinero del restaurante (que tiene DNI, nombre, país como datos de interés); de los platos se desea saber el código asignado al plato, el nombre y las calorías aproximadas. Puede interesar almacenar para algunos platos los ingredientes (indicando la cantidad) y el vino que se recomienda para un correcto disfrute. De cada vino disponible en el restaurante se quiere conocer su código (interno al restaurante), el tipo, el nombre y la añada. Por último y para poder controlar la despensa, de cada ingrediente es interesante almacenar un código, el nombre, el precio en mercado y las existencias que quedan (obligatorio)."

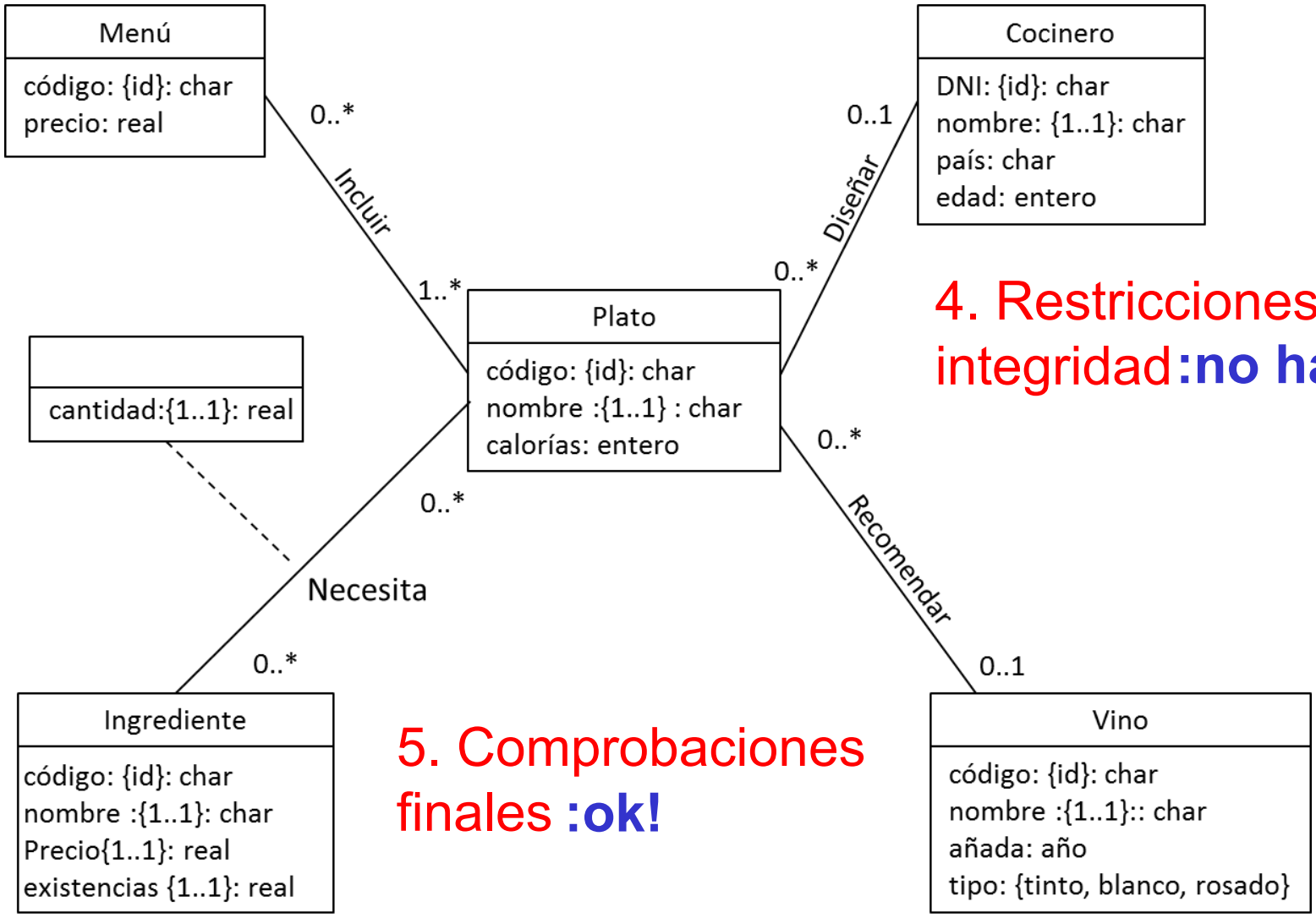
NO HAY

2.3.- Ejercicio 1: Restaurante

3. Identificar asociaciones entre clases

"Un restaurante desea que sus menús puedan ser consultados desde terminales y para ello ha decidido diseñar una base de datos relacional. En el restaurante se ofertan varios menús de los que se quiere saber el código, el precio y los platos que lo componen (al menos uno); cada plato puede haber sido diseñado como mucho por un cocinero del restaurante (que tiene DNI, nombre, país de origen y edad como datos de interés); de los platos se desea saber el código asignado al plato, el nombre y las calorías aproximadas; también puede interesar almacenar para algunos platos los ingredientes necesarios (indicando la cantidad) y el vino que se recomienda para su completo disfrute. De cada vino disponible en el restaurante se quiere conocer su código (interno al restaurante), el tipo, el nombre y la añada. Por último y para poder controlar la despensa, de cada ingrediente es interesante almacenar un código, el nombre, el precio en mercado y las existencias que quedan (obligatorio)."

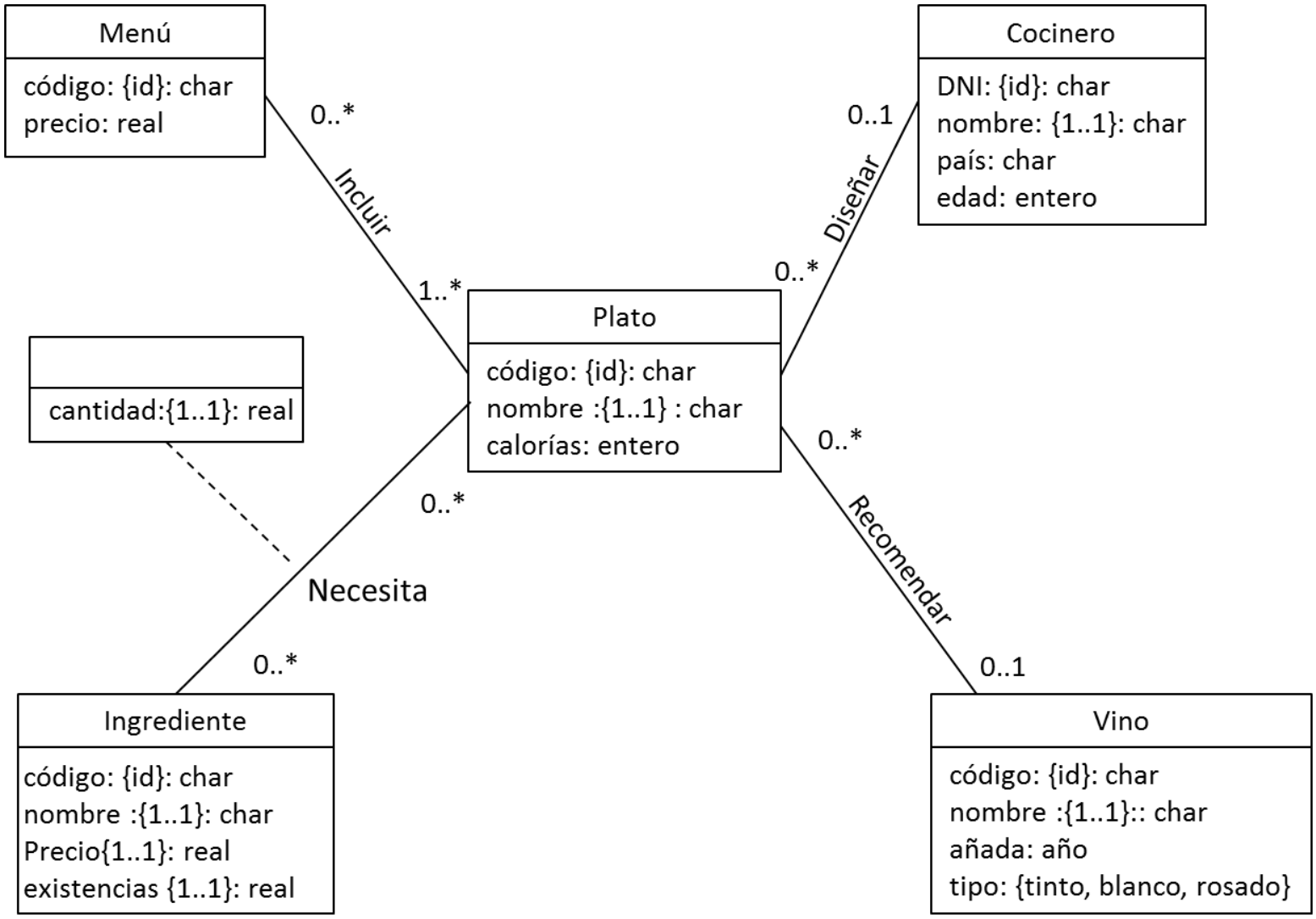
2.3.- Ejercicio 1: Restaurante



4. Restricciones de integridad: **no hay**

5. Comprobaciones finales: **ok!**

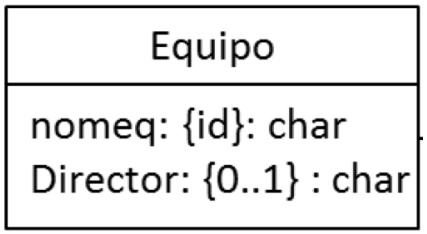
2.3.- Ejercicio 1: Restaurante

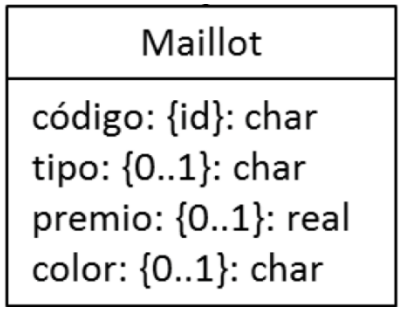


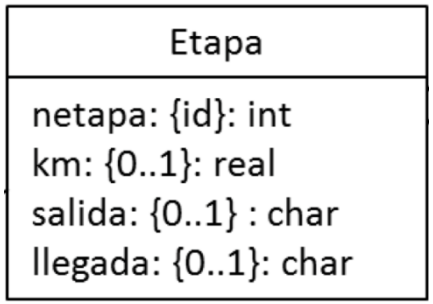
UD 4.2.- Diseño Conceptual

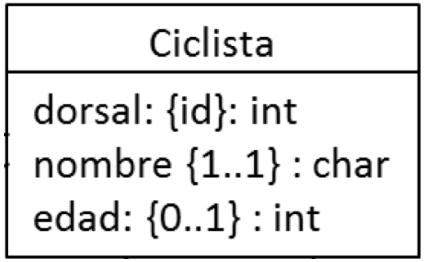
2.4.- Ejercicio 2: Ciclismo

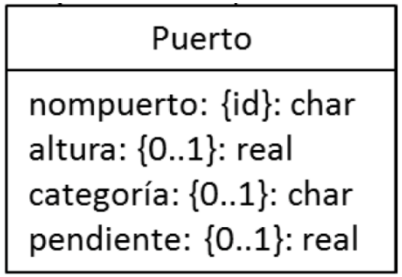
1. Identificar las clases con sus atributos
3. Identificar asociaciones entre clases



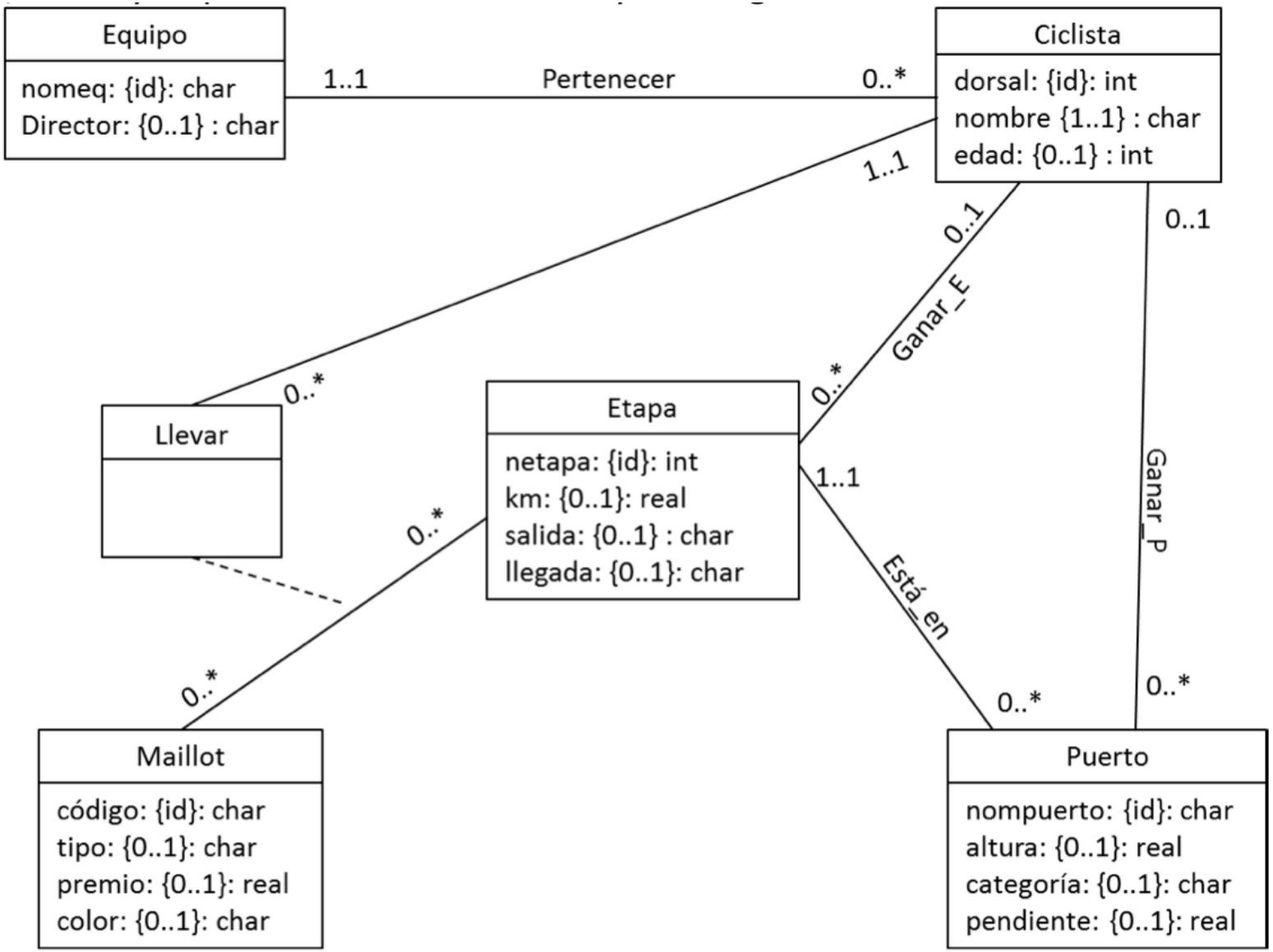




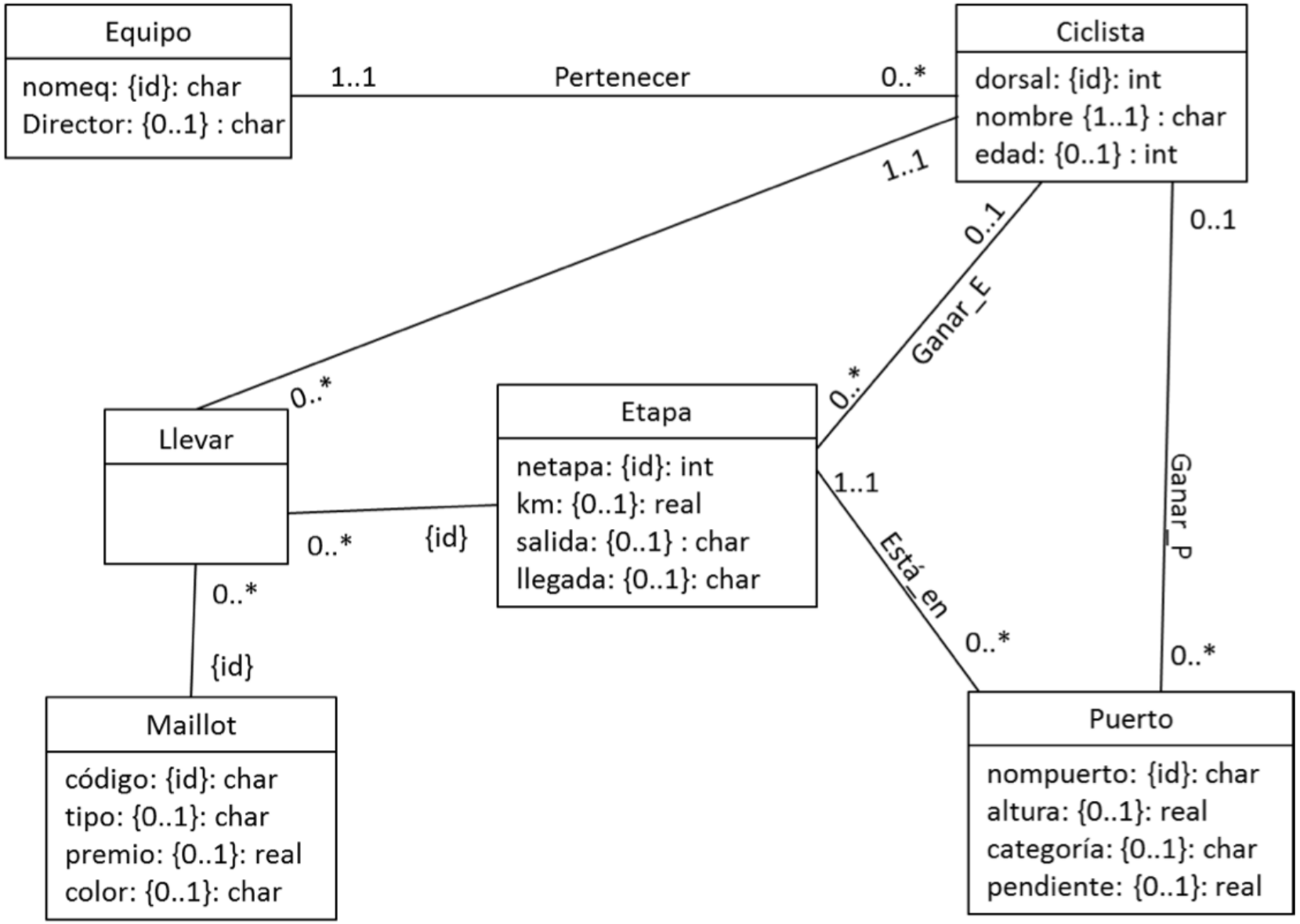




2.4.- Ejercicio 2: Ciclismo



2.4.- Ejercicio 2: Ciclismo



UD 4.2.- Diseño Conceptual

2.5.- Ejercicio 3: Compañía de seguros

Una compañía de seguros desea diseñar un sistema de información para gestionar las peritaciones de los vehículos accidentados a su cargo. Cada peritación se identifica con un número de referencia y se debe conocer necesariamente la fecha de realización, el perito asignado (código y nombre), los datos del taller donde se ha realizado (nombre del taller y domicilio) y del vehículo peritado. Cada vehículo, se codifica con un identificador secuencial. Para los vehículos matriculados además se almacena la matrícula, para los ciclomotores el número de placa municipal y para cualquier otro (por ejemplo bicicletas) un código interno (las matrículas, números de placa, y códigos internos en conjunto deben ser únicos). Además, interesa saber la marca, modelo, propietario y si está asegurado, el número de póliza, la compañía, el tipo y la fecha de caducidad (toda póliza siempre va ligada a un vehículo). Sólo se tiene información de los vehículos sobre los que se hacen peritaciones.

Se dispone de un catálogo de las diferentes partes de los vehículos codificadas con su correspondiente descripción (por ejemplo, XX1 Aleta delantera, XX2 Puerta Derecha, XX3 Faro Derecho, etc.). El resultado de la peritación consiste en una estimación de las partes del vehículo afectadas. Para cada parte afectada, se emite un diagnóstico en el que se indica si se ha de reparar o sustituir, así como el tiempo de mano de obra estimado. Además, se detallan los materiales a utilizar en la reparación de cada parte afectada. De estos materiales también se dispone de un catálogo con el código, descripción y precio.

2.5.- Ejercicio 3: Compañía de seguros

1. Identificar las clases con sus atributos

Una compañía de seguros desea diseñar un sistema de información para gestionar las peritaciones de los vehículos accidentados a su cargo. Cada peritación se identifica con un número de referencia y se debe conocer necesariamente la fecha de realización, el perito asignado (código y nombre), los datos del taller donde se ha realizado (nombre del taller y domicilio) y del vehículo peritado. Cada vehículo, se codifica con un identificador secuencial. Para los vehículos matriculados además se almacena la matrícula, para los ciclomotores el número de placa municipal y para cualquier otro (por ejemplo bicicletas) un código interno (las matrículas, números de placa, y códigos internos en conjunto deben ser únicos). Además, interesa saber la marca, modelo, propietario y si está asegurado, el número de póliza, la compañía, el tipo y la fecha de caducidad (toda póliza siempre va ligada a un vehículo). Sólo se tiene información de los vehículos sobre los que se hacen peritaciones.

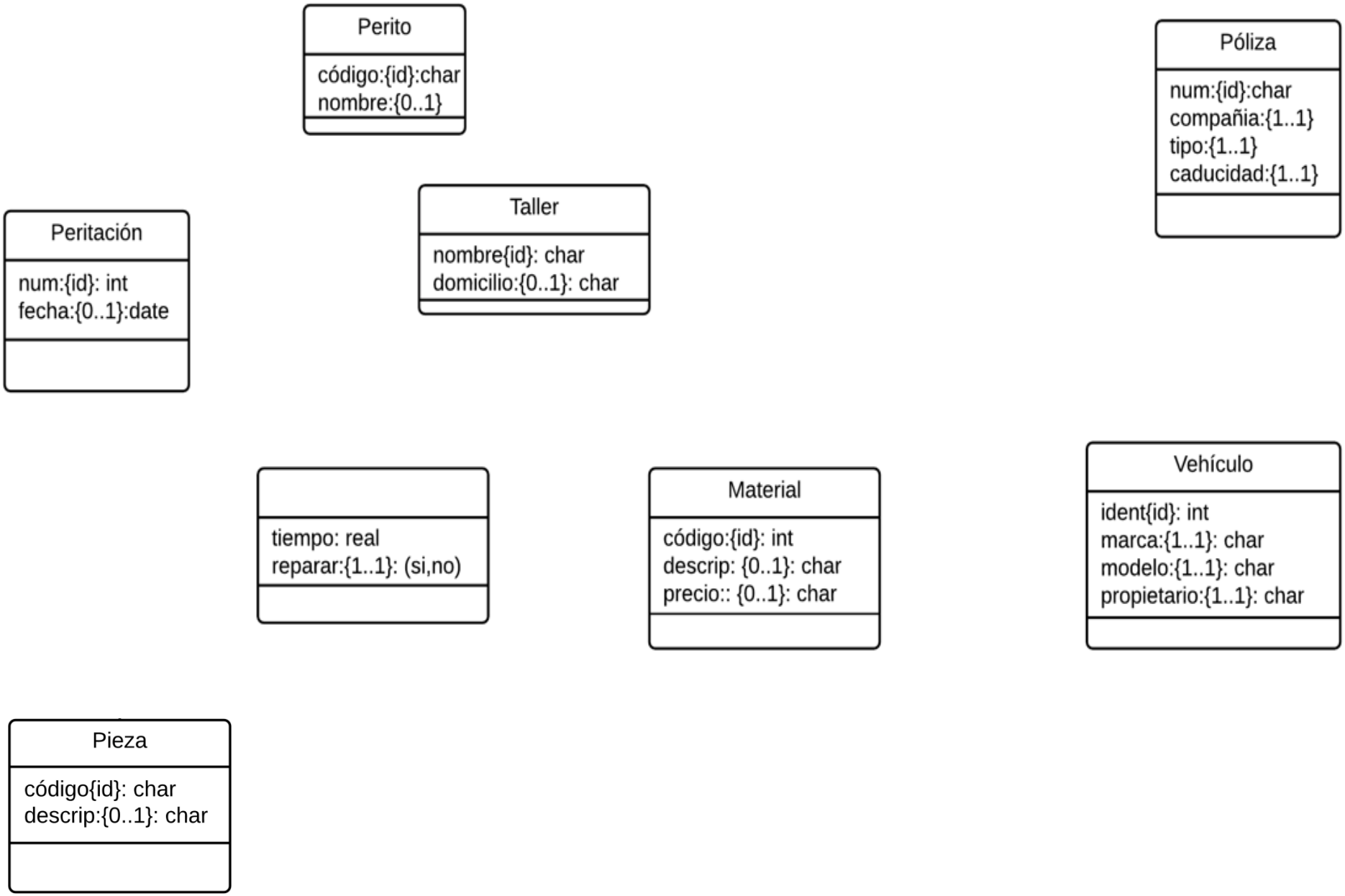
Se dispone de un catálogo de las diferentes partes de los vehículos codificadas con su correspondiente descripción (por ejemplo, XX1 Aleta delantera, XX2 Puerta Derecha, XX3 Faro Derecho, etc.). El resultado de la peritación consiste en una estimación de las partes del vehículo afectadas. Para cada parte afectada, se emite un diagnóstico en el que se indica si se ha de reparar o sustituir, así como el tiempo de mano de obra estimado. Además, se detallan los materiales a utilizar en la reparación de cada parte afectada. De estos materiales también se dispone de un catálogo con el código, descripción y precio.

2.5.- Ejercicio 3: Compañía de seguros

1. Identificar las clases con sus atributos

Una compañía de seguros desea diseñar un sistema de información para gestionar las peritaciones de los vehículos accidentados a su cargo. Cada peritación se identifica con un número de referencia y se debe conocer necesariamente la fecha de realización, el perito asignado (código y nombre), los datos del taller donde se ha realizado (nombre del taller y domicilio) y del vehículo peritado. Cada vehículo, se codifica con un identificador secuencial. Para los vehículos matriculados además se almacena la matrícula, para los ciclomotores el número de placa municipal y para cualquier otro (por ejemplo bicicletas) un código interno (las matrículas, números de placa, y códigos internos en conjunto deben ser únicos). Además, interesa saber la marca, modelo, propietario y si está asegurado, el número de póliza, la compañía, el tipo y la fecha de caducidad (toda póliza siempre va ligada a un vehículo). Sólo se tiene información de los vehículos sobre los que se hacen peritaciones.

Se dispone de un catálogo de las diferentes partes de los vehículos codificadas con su correspondiente descripción (por ejemplo, XX1 Aleta delantera, XX2 Puerta Derecha, XX3 Faro Derecho, etc.). El resultado de la peritación consiste en una estimación de las partes del vehículo afectadas. Para cada parte afectada, se emite un diagnóstico en el que se indica si se ha de reparar o sustituir, así como el tiempo de mano de obra estimado. Además, se detallan los materiales a utilizar en la reparación de cada parte afectada. De estos materiales también se dispone de un catálogo con el código, descripción y precio.

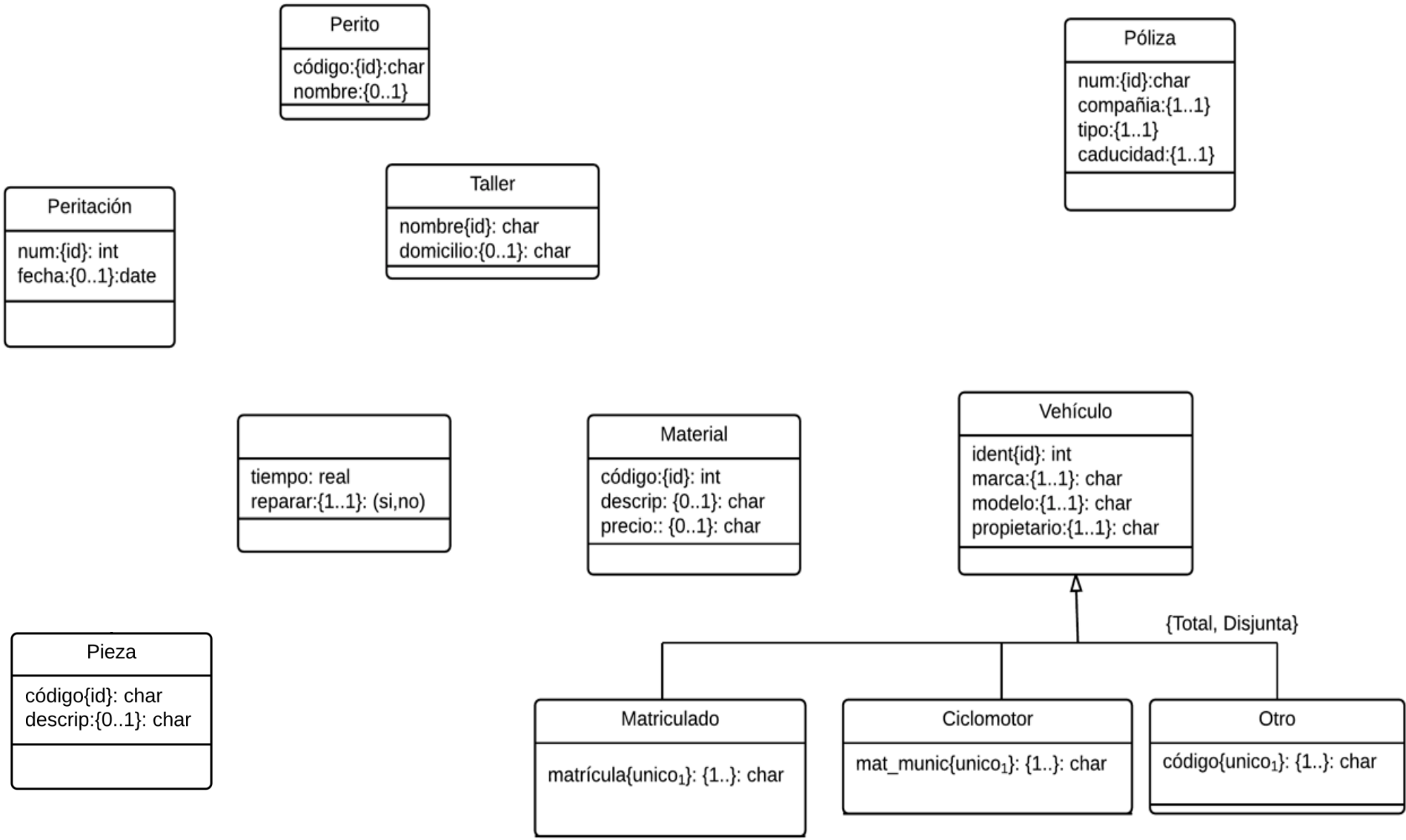

Perito
código:{id}:char
nombre:{0..1}
Póliza
num:{id}:char
compañia:{1..1}
tipo:{1..1}
caducidad:{1..1}
Peritación
num:{id}: int
fecha:{0..1}:date
Taller
nombre{id}: char
domicilio:{0..1}: char
tiempo: real
reparar:{1..1}: (si,no)
Material
código:{id}: int
descrip: {0..1}: char
precio:: {0..1}: char
Vehículo
ident{id}: int
marca:{1..1}: char
modelo:{1..1}: char
propietario:{1..1}: char
Pieza
código{id}: char
descrip:{0..1}: char

2.5.- Ejercicio 3: Compañía de seguros

2. Identificar generalizaciones/especializaciones

Una compañía de seguros desea diseñar un sistema de información para gestionar las peritaciones de los vehículos accidentados a su cargo. Cada peritación se identifica con un número de referencia y se debe conocer necesariamente la fecha de realización, el perito asignado (código y nombre), los datos del taller donde se ha realizado (nombre del taller y domicilio) y del vehículo peritado. Cada vehículo, se codifica con un identificador secuencial. Para los vehículos matriculados además se almacena la matrícula, para los ciclomotores el número de placa municipal y para cualquier otro (por ejemplo bicicletas) un código interno (las matrículas, números de placa, y códigos internos en conjunto deben ser únicos). Además, interesa saber la marca, modelo, propietario y si está asegurado, el número de póliza, la compañía, el tipo y la fecha de caducidad (toda póliza siempre va ligada a un vehículo). Sólo se tiene información de los vehículos sobre los que se hacen peritaciones.

Se dispone de un catálogo de las diferentes partes de los vehículos codificadas con su correspondiente descripción (por ejemplo, XX1 Aleta delantera, XX2 Puerta Derecha, XX3 Faro Derecho, etc.). El resultado de la peritación consiste en una estimación de las partes del vehículo afectadas. Para cada parte afectada, se emite un diagnóstico en el que se indica si se ha de reparar o sustituir, así como el tiempo de mano de obra estimado. Además, se detallan los materiales a utilizar en la reparación de cada parte afectada. De estos materiales también se dispone de un catálogo con el código, descripción y precio.

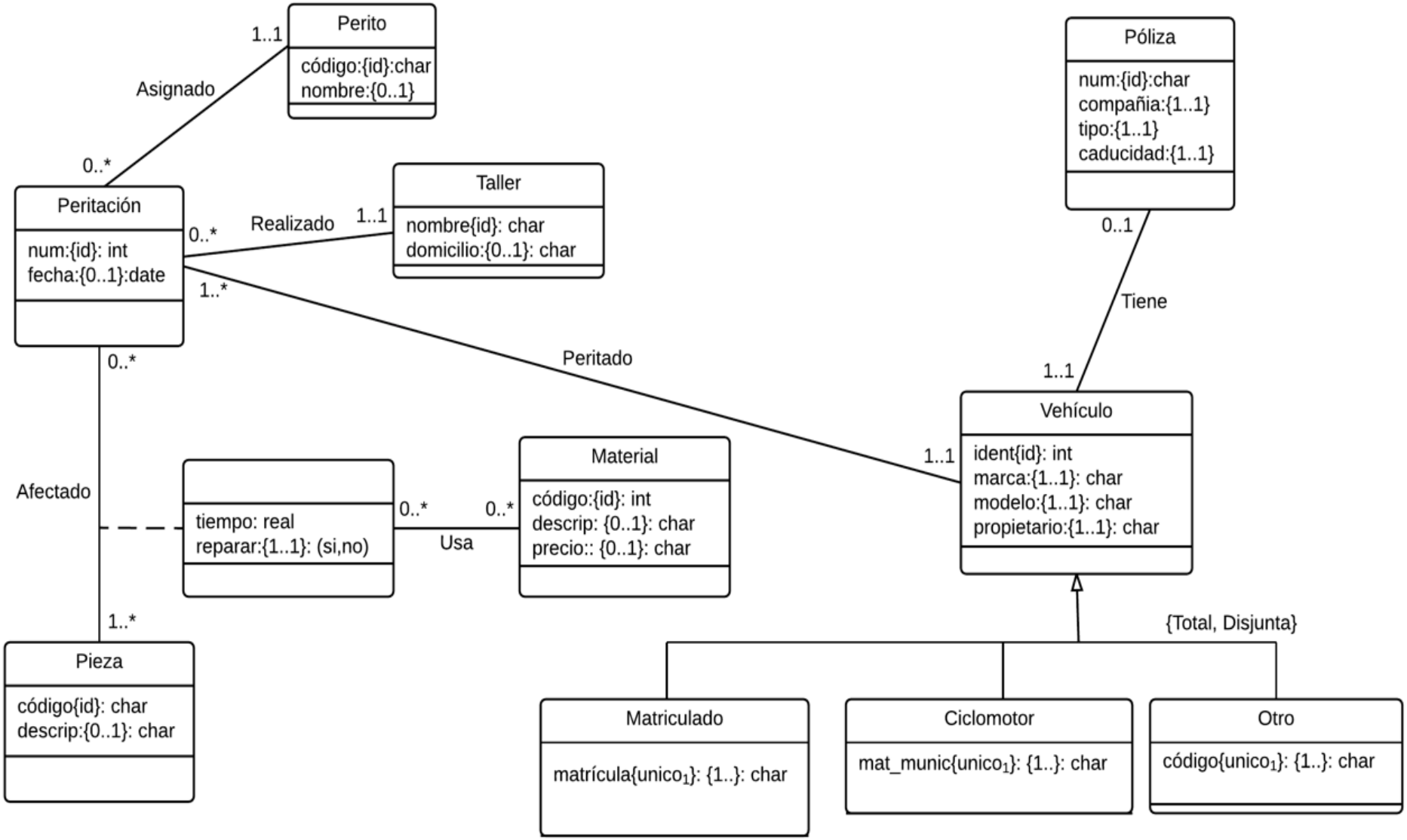

Perito
código:{id}:char
nombre:{0..1}
Póliza
num:{id}:char
compañia:{1..1}
tipo:{1..1}
caducidad:{1..1}
Taller
nombre{id}: char
domicilio:{0..1}: char
Peritación
num:{id}: int
fecha:{0..1}:date
tiempo: real
reparar:{1..1}: (si,no)
Material
código:{id}: int
descrip: {0..1}: char
precio:: {0..1}: char
Vehículo
ident{id}: int
marca:{1..1}: char
modelo:{1..1}: char
propietario:{1..1}: char
{Total, Disjunta}
Pieza
código{id}: char
descrip:{0..1}: char
Matriculado
matrícula{unico1}: {1..}: char
Ciclomotor
mat_munic{unico1}: {1..}: char
Otro
código{unico1}: {1..}: char

2.5.- Ejercicio 3: Compañía de seguros

3. Identificar asociaciones entre clases

Una compañía de seguros desea diseñar un sistema de información para gestionar las peritaciones de los vehículos accidentados a su cargo. Cada peritación se identifica con un número de referencia y se debe conocer necesariamente la fecha de realización, el perito asignado (código y nombre), los datos del taller donde se ha realizado (nombre del taller y domicilio) y del vehículo peritado. Cada vehículo, se codifica con un identificador secuencial. Para los vehículos matriculados además se almacena la matrícula, para los ciclomotores el número de placa municipal y para cualquier otro (por ejemplo bicicletas) un código interno (las matrículas, números de placa, y códigos internos en conjunto deben ser únicos). Además, interesa saber la marca, modelo, propietario y si está asegurado, el número de póliza, la compañía, el tipo y la fecha de caducidad (toda póliza siempre va ligada a un vehículo). Sólo se tiene información de los vehículos sobre los que se hacen peritaciones.

Se dispone de un catálogo de las diferentes partes de los vehículos codificadas con su correspondiente descripción (por ejemplo, XX1 Aleta delantera, XX2 Puerta Derecha, XX3 Faro Derecho, etc.). El resultado de la peritación consiste en una estimación de las partes del vehículo afectadas. Para cada parte afectada, se emite un diagnóstico en el que se indica si se ha de reparar o sustituir, así como el tiempo de mano de obra estimado. Además, se detallan los materiales a utilizar en la reparación de cada parte afectada. De estos materiales también se dispone de un catálogo con el código, descripción y precio.


Perito
código:{id}:char
nombre:{0..1}
1..1
Asignado
0..*
Póliza
num:{id}:char
compañia:{1..1}
tipo:{1..1}
caducidad:{1..1}
Peritación
num:{id}: int
fecha:{0..1}:date
0..*
Realizado
1..1
Taller
nombre{id}: char
domicilio:{0..1}: char
0..1
Tiene
1..1
1..*
Peritado
0..*
Vehículo
ident{id}: int
marca:{1..1}: char
modelo:{1..1}: char
propietario:{1..1}: char
1..1
Material
código:{id}: int
descrip: {0..1}: char
precio:: {0..1}: char
Afectado
tiempo: real
reparar:{1..1}: (si,no)
0..*
0..*
Usa
1..*
Pieza
código{id}: char
descrip:{0..1}: char
{Total, Disjunta}
Matriculado
matrícula{unico1}: {1..}: char
Ciclomotor
mat_munic{unico1}: {1..}: char
Otro
código{unico1}: {1..}: char