



INGENIERÍA DEL SOFTWARE I

Tema 8

Estructura del Sistema
(en desarrollo OO)

Univ. Cantabria – Fac. de Ciencias

Carlos Blanco, Francisco Ruiz



Objetivos del Tema

- Conocer en detalle los **elementos** que permiten representar la **estructura de un sistema**.
- Aprender a realizar **diagramas de clases y de objetos de UML 2**.
- Aprender a **modelar** con ellos diferentes **aspectos estructurales**, del dominio del problema (requisitos-análisis) y del dominio de la solución (diseño del sistema).
- Aprender a usar los **mecanismos** que permiten **extender y particularizar UML**.



Contenido

1. Introducción

2. Elementos Estructurales

- Clase
- Atributo
- Operación
- Responsabilidad
- Relaciones
 - Dependencia
 - Generalización
 - Asociación
 - Restricciones entre Relaciones
- Clases Especiales
- Otros Clasificadores

3. Interfaces

- Realización

4. Diagramas de Clases

- Consejos

5. Objetos

- Estado
- Diagramas de Objetos
- Consejos

6. Modelado

- Vocabulario del Sistema
- Distribución de Responsabilidades
- Semántica de una Clase
- Colaboraciones
- Esquemas de Datos
- Redes de Relaciones
- Líneas de Separación
- Instancias

7. Mecanismos de Extensión

- Estereotipos
- Valores Etiquetados
- Restricciones
- Perfiles



Bibliografía

- Básica

- Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado. 2ª edición.
 - Caps. 4-6, 8-11 y 13-14.

- Complementaria

- Rumbaugh, Jacobson y Booch (2007): El Lenguaje Unificado de Modelado. Manual de Referencia. 2ª edición.
 - Cap. 4.
- Hamilton y Miles (2006): Learning UML 2.0.
 - Caps. 4-6.



Introducción

1. Introducción

● Modelado Estructural

- Se describen los **tipos de objetos** de un sistema y las **relaciones estáticas** que existen entre ellos.
 - Clases
 - Interfaces
 - Relaciones de dependencia, realización, generalización y asociación (agregación, composición)
- Se suele representar mediante diagramas de clases, y opcionalmente, diagramas de objetos.



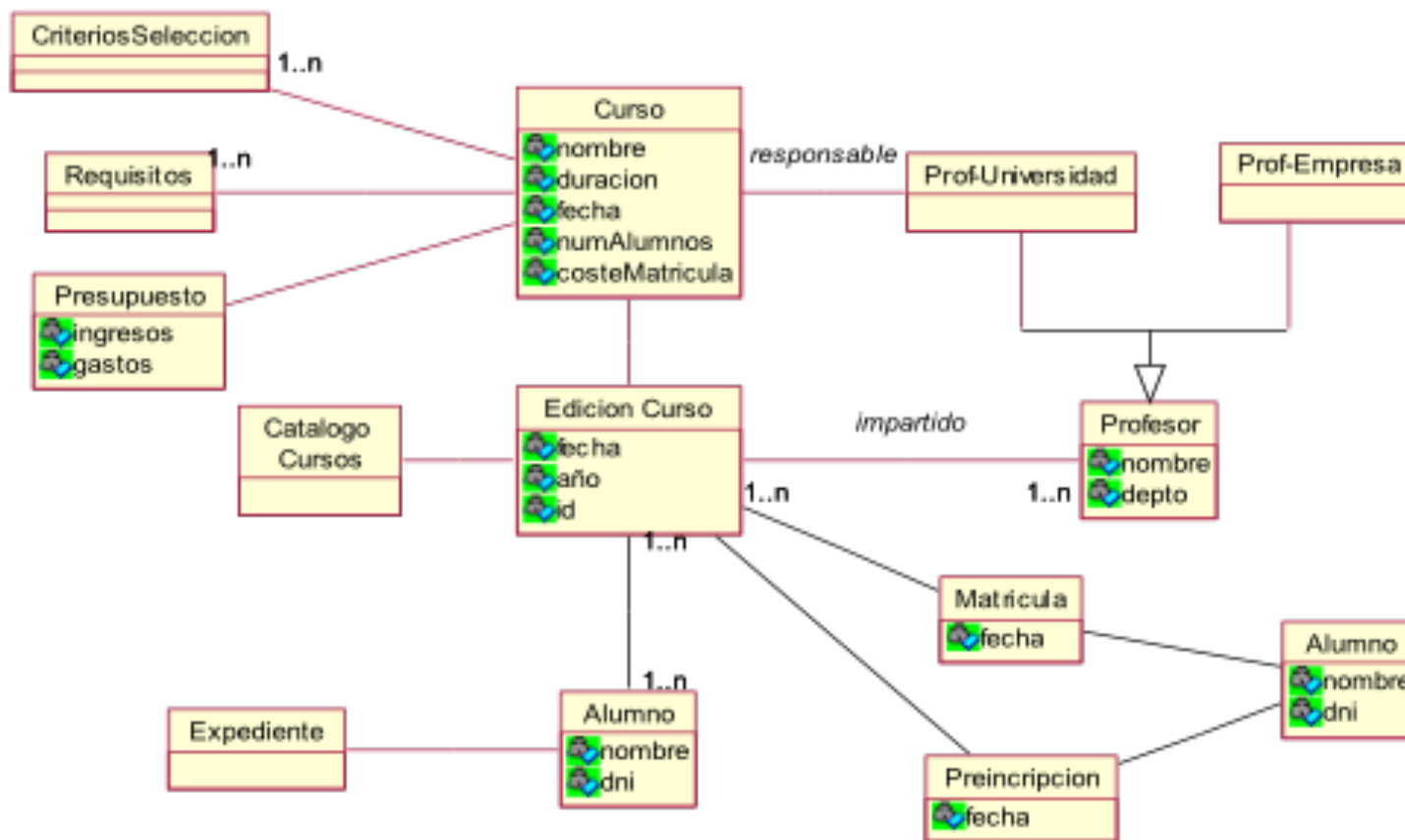
Introducción

- El **Modelado Estructural** se puede realizar desde diferentes puntos de vista resultando diversos tipos de modelos del sistema:
 - **Modelo Conceptual**
 - Conceptos del dominio del problema: atributos, restricciones y relaciones entre ellos.
 - **Modelo de Análisis**
 - Clases que corresponden a conceptos del dominio.
 - Atributos y operaciones
 - **Modelo de Diseño**
 - Incluye clases que corresponden a decisiones del diseño.
 - **Modelo de Implementación**
 - Clases que corresponden a una tecnología de implementación (lenguaje de programación).



Introducción

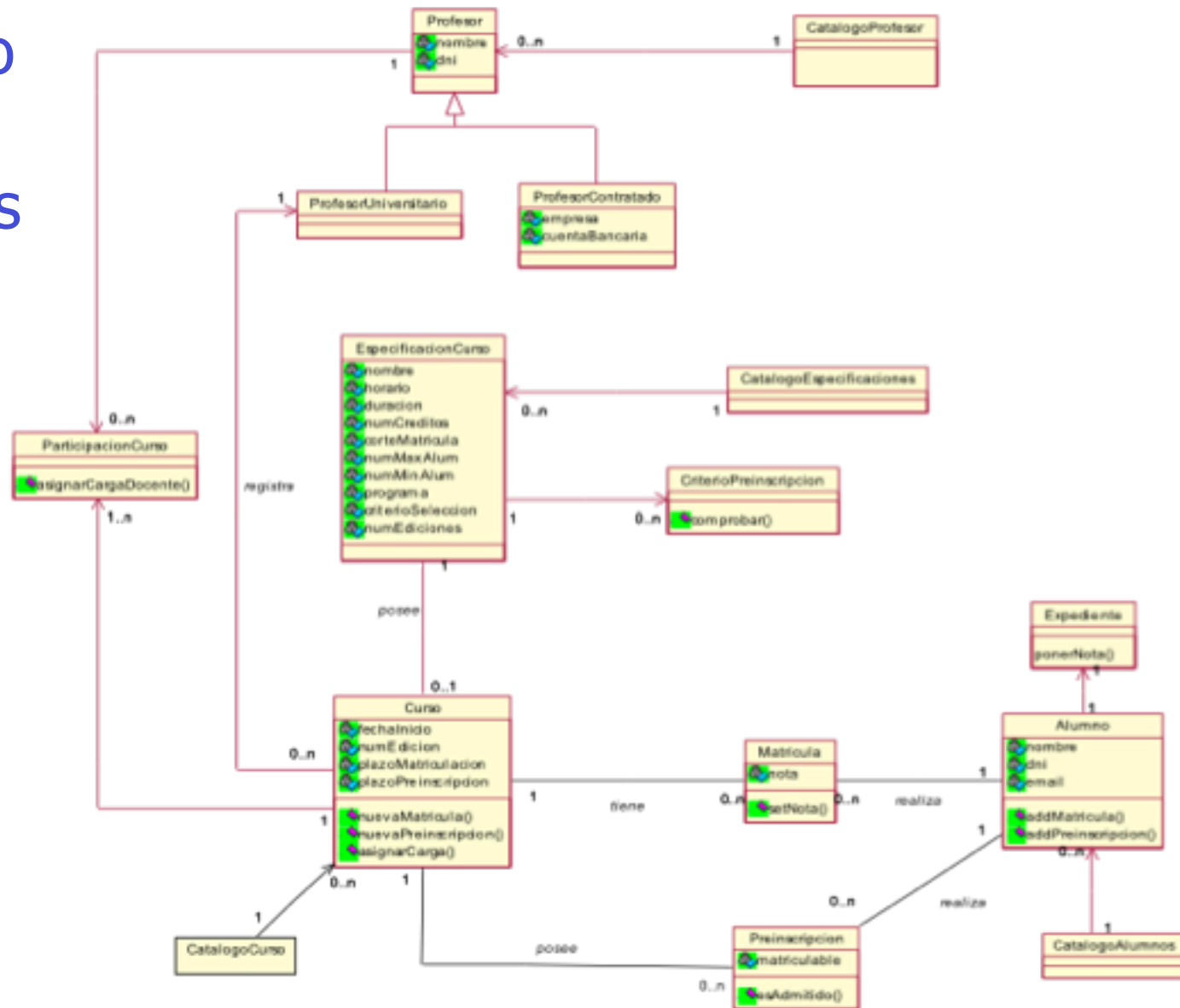
- Modelo Conceptual





Introducción

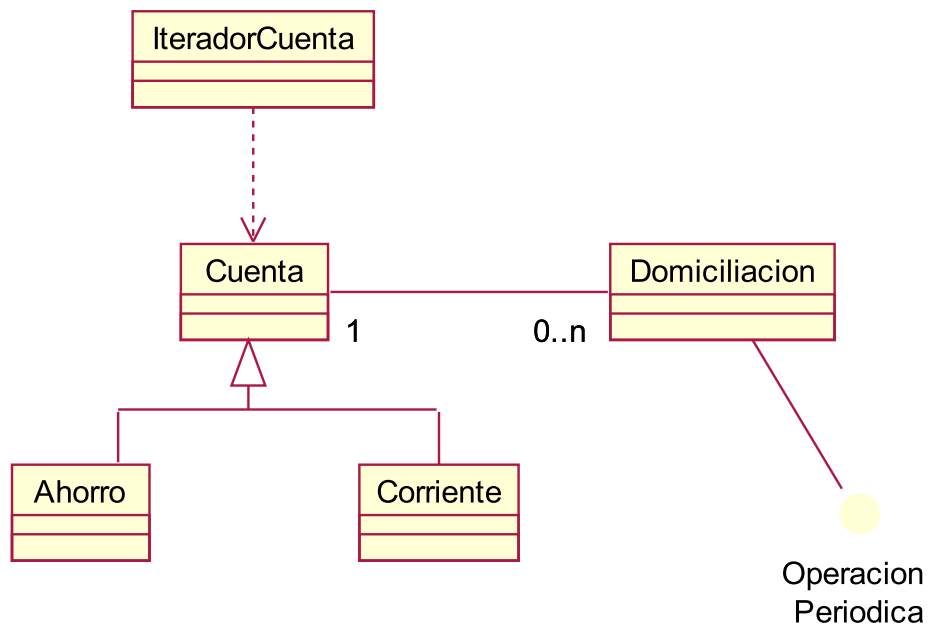
- Modelo de Análisis





Introducción

- Modelo de Diseño





Elementos Estructurales - Clase

2. Elementos Estructurales

● Clases

- El mundo real puede ser visto desde abstracciones diferentes (subjetividad).
- Los principales **mecanismos de abstracción** para pensar sobre el mundo son:
 - **Clasificación / Instanciación**
 - **Composición / Descomposición**
 - **Agrupación / Individualización**
 - **Especialización / Generalización**
- La **clasificación** es uno de los mecanismos de abstracción más utilizados.
 - Tanto en la vida cotidiana



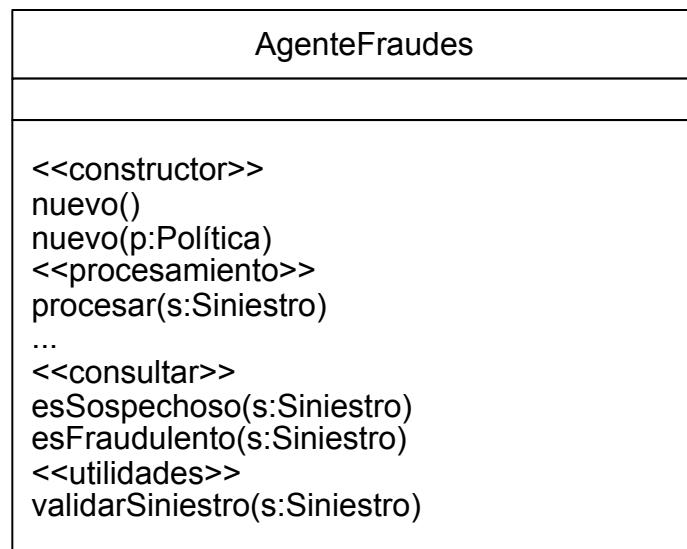
Elementos Estructurales - Clase

- Las **Clases** son la principal manera de aplicar **clasificación en UML**.
 - Sirven para identificar las “cosas” importantes desde una visión particular.
 - Constituyen el vocabulario del sistema que se modela.
 - Cada una de ellas tiene ciertas propiedades y un comportamiento.
- Una clase define el ámbito de definición de un **conjunto de objetos** =>
 - Cada objeto pertenece a una clase.
 - Los objetos se crean por instanciación de las clases.



Elementos Estructurales - Clase

- Cuando se modela no hay que contemplar todo el detalle de las clases (**abstracción de modelado**):
 - Al dibujar una clase no hay que mostrar todos los atributos y operaciones.
 - Se especifica con puntos suspensivos (...) que hay más atributos u operaciones.
 - Se utilizan estereotipos como categoría descriptiva para organizar atributos y operaciones.





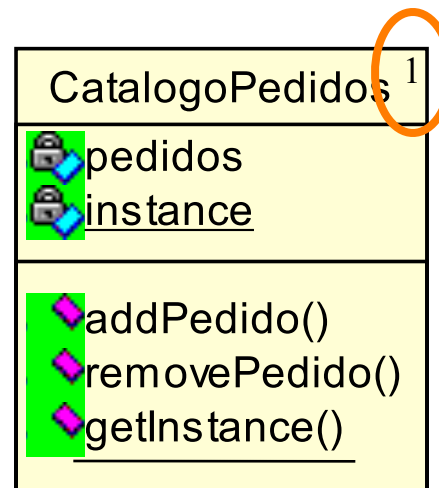
Elementos Estructurales - Clase

- Las características básicas de las clases UML
 - Nombre, Atributos, Operaciones
- Se complementan con otras **características avanzadas**:
 - Visibilidad de atributos y operaciones.
 - Alcance de atributos.
 - Multiplicidad de clases y atributos.
 - Valor inicial y modificabilidad de atributos.
 - Tipo de las operaciones.
- Sirven para refinar la definición de las clases conforme avanza el proyecto.



Elementos Estructurales - Clase

- A veces se desea restringir el **número de instancias** (**multiplicidad**) que puede tener una clase:
 - 1 => Clase unitaria o singleton.
 - k





Elementos Estructurales - Atributo

- **Atributo**
- Además de su nombre (obligatorio), para un **atributo** (property en UML) se pueden especificar otras características, según la siguiente sintaxis:

```
[<visibilidad>] [ '/' ] <nombre> [ ':' <tipo> ]  
[ [ ' <multiplicidad> ' ] ] [ '=' <valor inicial> ]  
[ { ' <modificador> [ ',' <modificador> ] * ' } ]
```

- donde

- '/' significa que es un atributo derivado
- <modificador> representa un modificador que cambia la naturaleza del atributo.



Elementos Estructurales - Atributo

● Ejemplos de Atributos:

- origen nombre
- + origen visibilidad y nombre
- origen : Punto nombre y tipo
- nombre : String [0..1] nombre, tipo y multiplicidad
- origen : Punto = (0,0) nombre, tipo y valor inicial
- id : Integer {readOnly} nombre, tipo y modificador
- / edad : Integer nombre y tipo (atributo derivado)



Elementos Estructurales - Atributo

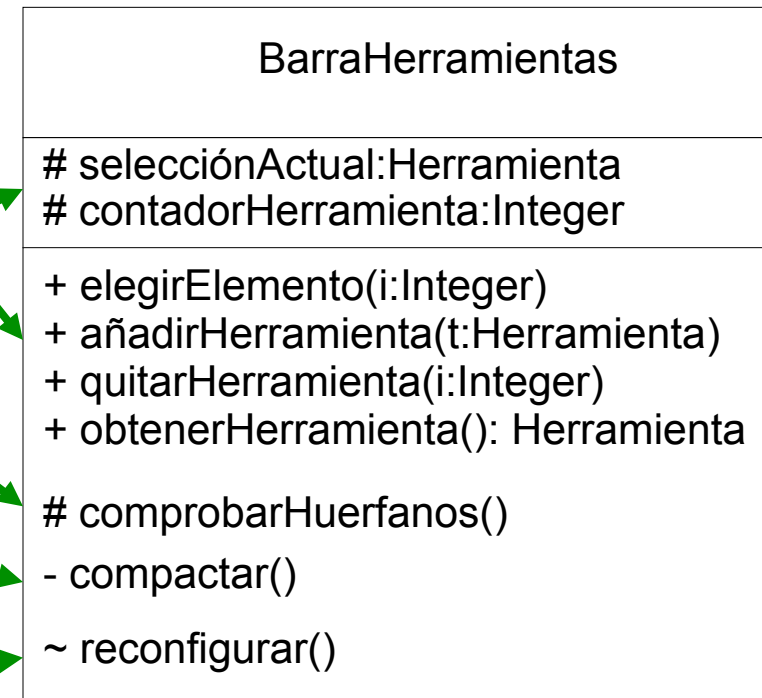
- La **Visibilidad** es aplicable a atributos y operaciones.
 - Sirve para **ocultar los detalles** de implementación (encapsulación) y mostrar sólo aquellas características necesarias para llevar a cabo las responsabilidades de un clasificador.
 - Esta ocultación de información es esencial para construir sistemas sólidos y flexibles.



Elementos Estructurales - Atributo

- UML 2 tiene cuatro **niveles de visibilidad**:

- **public (+)**: Cualquier clasificador externo puede utilizar la característica (opción por defecto).
- **protected (#)**: Sólo los descendientes del clasificador pueden usarla.
- **private (-)**: Sólo el propio clasificador puede utilizarla.
- **package (~)**: Sólo los clasificadores declarados en el mismo paquete pueden utilizarla.



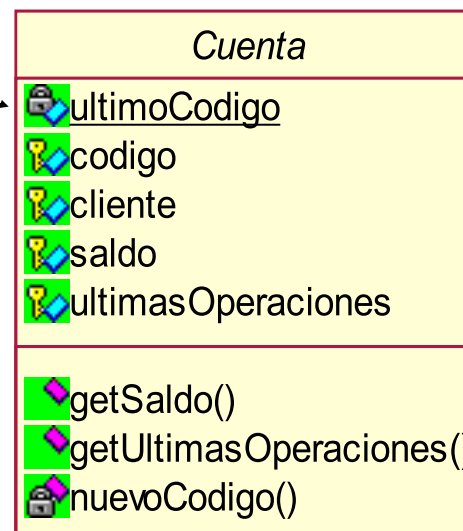


Elementos Estructurales - Atributo

- Una característica (atributo u operación) de una clase (o clasificador en general) puede ser estática o dinámica:
 - **Dinámica** (IsStatic=false)
 - Cada instancia del clasificador tiene su propio valor o instancia para la característica.
 - Opción por defecto.
 - **Estática** (IsStatic=true)
 - Hay un único valor o instancia para todas las instancias del clasificador.
 - Se muestra subrayando el nombre.

En UML 1.x era la propiedad

Alcance (instance vs classifier)





Elementos Estructurales - Atributo

- En el caso de los atributos, la **Multiplicidad** indica el número de valores simultáneos que pueden existir para cada instancia de la clase.
 - La opción por defecto es un solo valor.
 - En otro caso, se indica señalando entre corchetes el número mínimo (lower) y máximo (upper) de valores posibles:

```
` [ '[' <mínimo> '..' ] <máximo> [ '{' <orden> [ ',' <unicidad> ] '}' ] ` ] `
```
 - Donde
 - <mínimo> es un valor entero
 - <máximo> es un asterisco (*) o un valor entero.
 - <orden> = { 'ordered' | 'unordered' }
 - <unicidad> = { 'unique' | 'nonunique' }



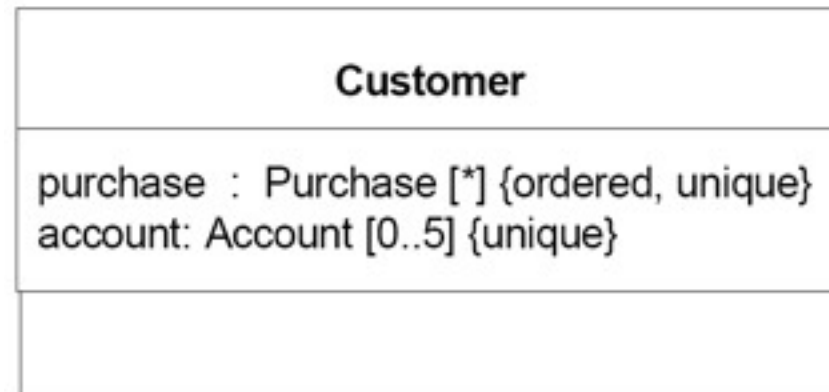
Elementos Estructurales - Atributo

- Ejemplos de Multiplicidad de Atributos:

- [0..1]
- [1..3]
- [1..*]
- [*]

- En un atributo multivaluado (máximo > 1) se pueden incluir restricciones de:

- Orden (ordered)
- Unicidad (unique)





Elementos Estructurales - Atributo

- Los **modificadores** que se pueden aplicar a un **atributo** en UML 2 son:
 - **readOnly**: es de sólo lectura (no modificable)
 - **union**: es una unión derivada de sus subconjuntos (subsets).
 - **subsets <atributo2>**: es un subconjunto de otro atributo
 - **redefines <atributo2>**: redefine por herencia otro atributo.
 - **ordered**: los valores simultáneos de un atributo multivaluado están ordenados secuencialmente.
 - **unique**: no puede haber duplicados en un atributo multivaluado.
 - **<restricción de atributo>**: expresión que especifica una restricción a nivel de atributo.



Elementos Estructurales - Atributo

- Combinando los **modificadores Ordered y Unique** es posible crear **atributos** que son **colecciones** de diferentes clases:

Unique	Ordered	Tipo de Colección
true	false	Set (conjunto)
true	true	SetOrdered (conjunto ordenado)
false	false	Bag (bolsa)
false	true	Sequence (secuencia)



Elementos Estructurales - Operación

- **Operación**
- Es una característica de comportamiento (behavioral feature) de un clasificador que especifica información para invocar un comportamiento asociado al clasificador.
- UML 2 distingue entre **Operación y Método**:
 - Una operación especifica un servicio que se puede requerir de cualquier objeto de una clase.
 - Un método es una implementación de una operación.
 - Cada operación (no abstracta) tiene un método con el algoritmo ejecutable.
 - En una jerarquía de herencia puede haber varios métodos para la misma operación (polimorfismo).



Elementos Estructurales - Operación

- Sintaxis completa de una **Operación**:

[<visibilidad>] <nombre> '(' [<lista de parámetros>]')'
[':' [<tipo de retorno>]
'{' <modificador> [','<modificador>]*'}']

- donde

- <visibilidad> es similar a la de los atributos.
- <lista de parámetros> es una lista separada por comas.
- <tipo de retorno> es el tipo de datos del valor devuelto.
- <modificador> representa un modificador que cambia la naturaleza de la operación.

- Las operaciones pueden ser dinámicas o estáticas, igual que los atributos.



Elementos Estructurales - Operación

- Sintaxis para cada **parámetro** de una **Operación**:

[<dirección>] <nombre> ':' <tipo> ['['<multiplicidad>']']
['='<valor por defecto>]
['{' <modificador> [','<modificador>] '*' }']

- donde

- <dirección> indica si es de entrada (in), salida (out), ambos (inout) o se devuelve al llamador como retorno (return). La opción por defecto es in.
- <tipo> es el tipo de dato del parámetro.
- <multiplicidad> es similar a la de los atributos.
- <modificador> representa un modificador que afecta a la naturaleza del parámetro.



Elementos Estructurales - Operación

- Ejemplos de Operaciones:

- mostrar nombre
- + mostrar visibilidad y nombre
- set (n:Nombre, s:String) nombre y parámetros
- obtenerID(): Integer nombre y tipo de retorno
- saldo() {query} nombre y modificador



Elementos Estructurales - Operación

- Los **modificadores** que se pueden aplicar a una **operación** en UML 2 son:
 - **redefines <operacion2>**: redefine por herencia otra operación.
 - **query**: la operación no cambia el estado del sistema (consulta).
 - **ordered**: los valores simultáneos del parámetro de retorno (multivaluado) están ordenados secuencialmente.
 - **unique**: no puede haber duplicados en los valores simultáneos del parámetro de retorno (multivaluado).
 - **<restricción de operación>**: expresión que especifica una restricción a nivel de operación.



Elementos Estructurales - Operación

- En la especificación de una **operación** se puede indicar también su **semántica de concurrencia** (Call Concurrency Kind) como un modificador adicional:
 - **Sequential**: No se permite concurrencia.
 - **Guarded**: Se permiten varias instancias de ejecución de la operación, pero solo una se puede iniciar mientras las demás quedan bloqueadas hasta que la que está ejecutándose concluye.
 - **Concurrent**: Se permite concurrencia, es decir, varias instancias de ejecución de la operación a la vez.



Elementos Estructurales - Responsabilidad

- **Responsabilidad**
- Es un contrato u obligación de una clase (a nivel de análisis).
 - Ejemplo: Una clase pared es responsable de saber su altura, anchura y grosor.
 - Al ir refinando los modelos, las responsabilidades (texto libre) se traducen en el conjunto de atributos y operaciones que mejor satisfacen las responsabilidades de la clases.

AgenteDeFraudes
<p>Responsabilidades</p> <ul style="list-style-type: none">- Determinar el riesgo de un siniestro de un cliente- Gestionar criterios de fraude específicos para cada cliente



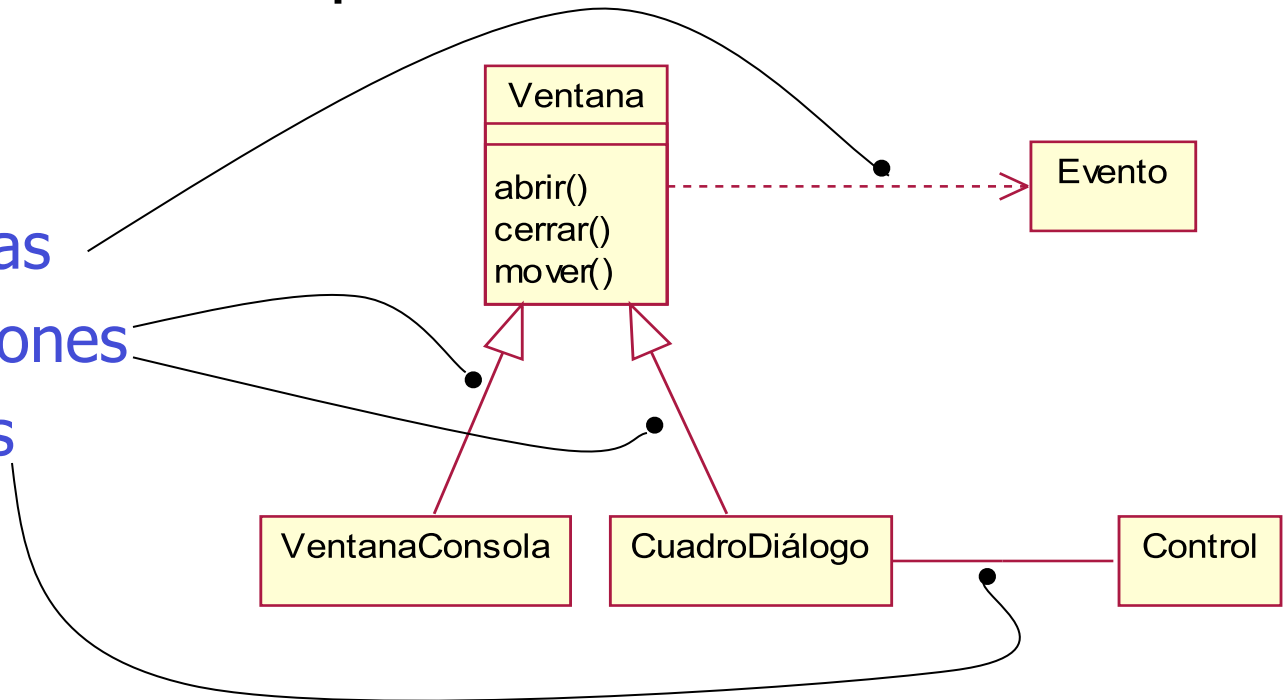
Elementos Estructurales - Relaciones

- **Relaciones**
- En el **modelado estructural** en **UML** hay tres clases de relaciones importantes:

■ Dependencias

■ Generalizaciones

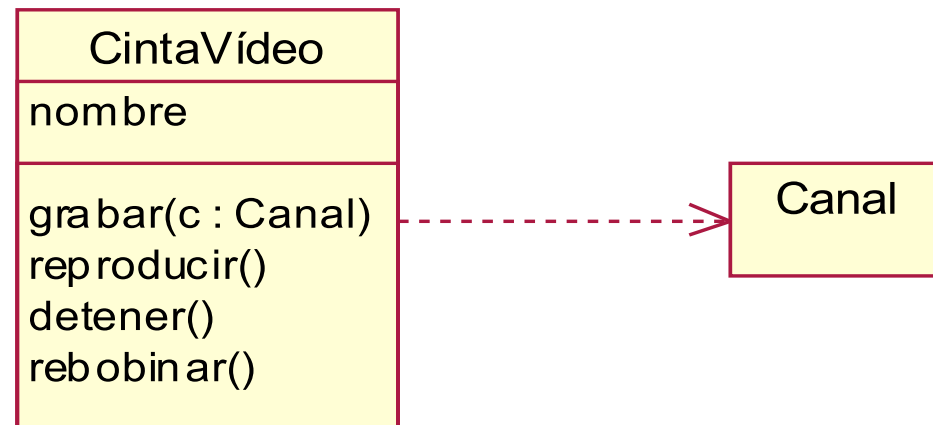
■ Asociaciones





Elementos Estructurales – Relaciones de Dependencia

- Un tipo común de **dependencia** es la relación de dos clases cuando una de ellas utiliza a la otra como parámetro de una operación.
 - Se modela mediante una dependencia que va de la clase que tiene la operación hasta la clase que es usada como parámetro:





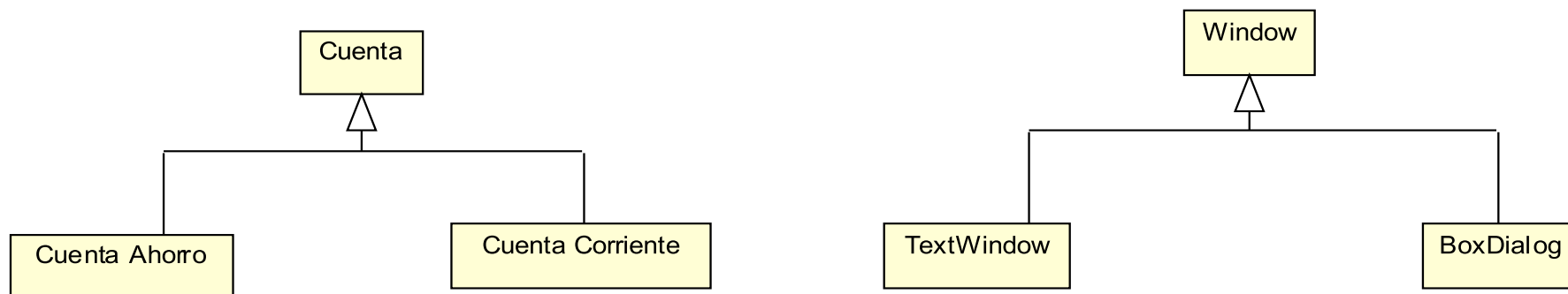
Elementos Estructurales – Relaciones de Dependencia

- Para precisar su naturaleza, **UML** incluye diversos **estereotipos de dependencia** entre clases y objetos:
 - **bind** - para clases plantilla; el origen de la dependencia instancia al destino.
 - **derive** - el origen se puede calcular a partir del destino.
 - **permit** - el origen tiene visibilidad especial en el destino.
 - **instanceOf** - el objeto (origen) es una instancia del clasificador (destino).
 - **instantiate** - el origen crea instancias del destino.
 - **powertype** - el destino es un supratipo (powertype) del origen. Un powertype es un clasificador cuyos objetos son todos los hijos posibles de un padre dado.
 - **refine** - el origen está más detallado que el destino.
 - **use** - la semántica del origen depende de la parte pública del destino.



Elementos Estructurales – Relaciones de Generalización

- La **Generalización** establece una relación con semántica “es-un-tipo-de” entre la clase hija (subclase) y la clase padre (superclase).
- En el caso de un modelo de **diseño o implementación** denota una relación de **herencia**.





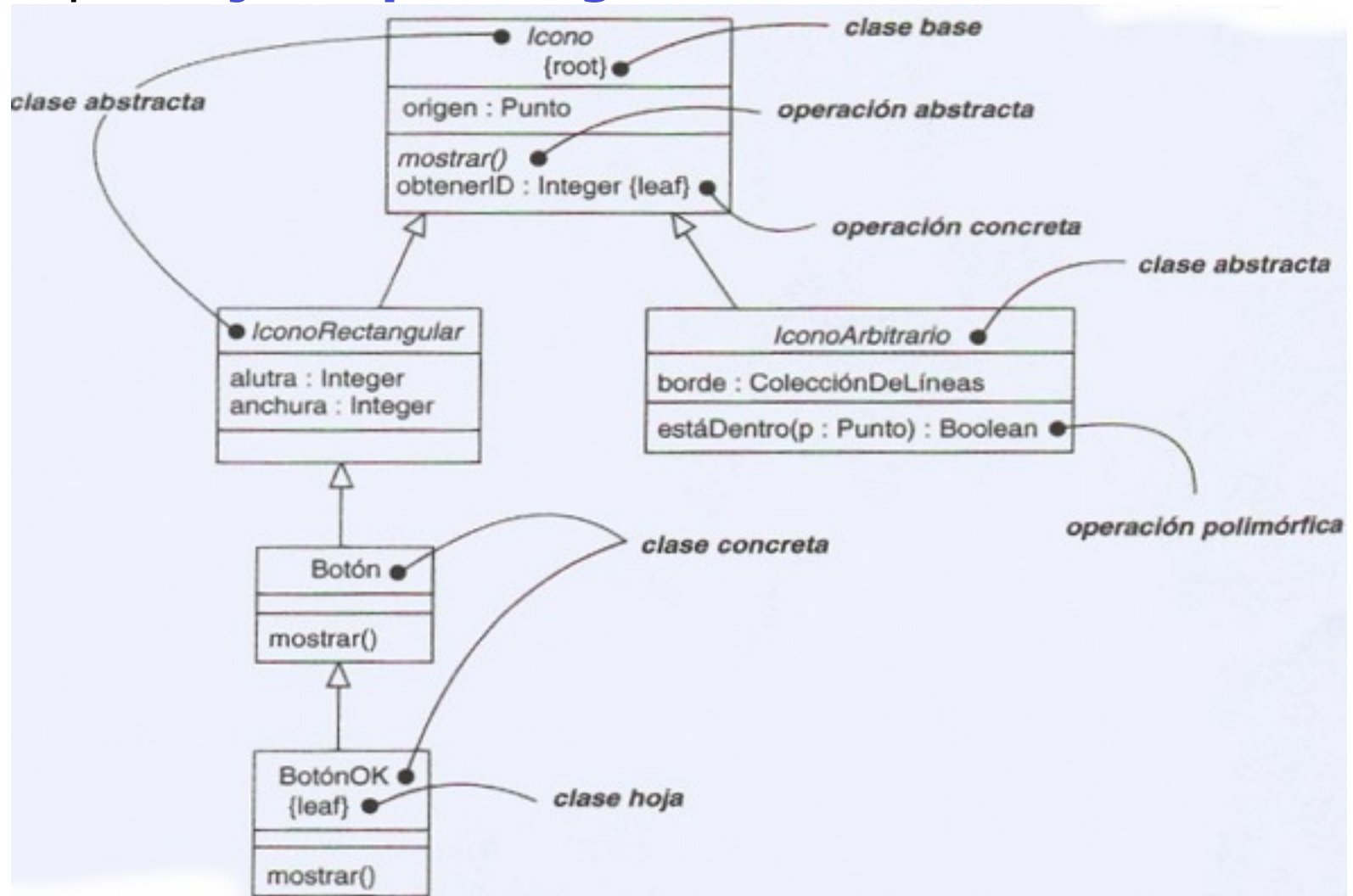
Elementos Estructurales – Relaciones de Generalización

- En las **jerarquías de generalización** existen **clases**
 - **Abstractas**: No tienen instancias directas.
 - Notación: Nombre en cursiva.
 - **Hoja**: No pueden tener hijas. [UML 2 no lo tiene]
 - Notación: Restricción {leaf} en el nombre de la clase.
 - **Raíz**: No tienen padres.
 - Notación: Restricción {root} en el nombre de la clase.
- Y **operaciones**
 - **Polimórficas**: Redefinen su comportamiento en varias clases hijas. Son incompletas y necesitan la redefinición en las hijas.
 - **Abstractas**: Necesitan de un complemento en las clases hijas.
 - Notación: Signatura en cursiva.
 - **Hoja**: No pueden ser redefinidas en clases hijas.
 - Notación: Modificador {leaf}.



Elementos Estructurales – Relaciones de Generalización

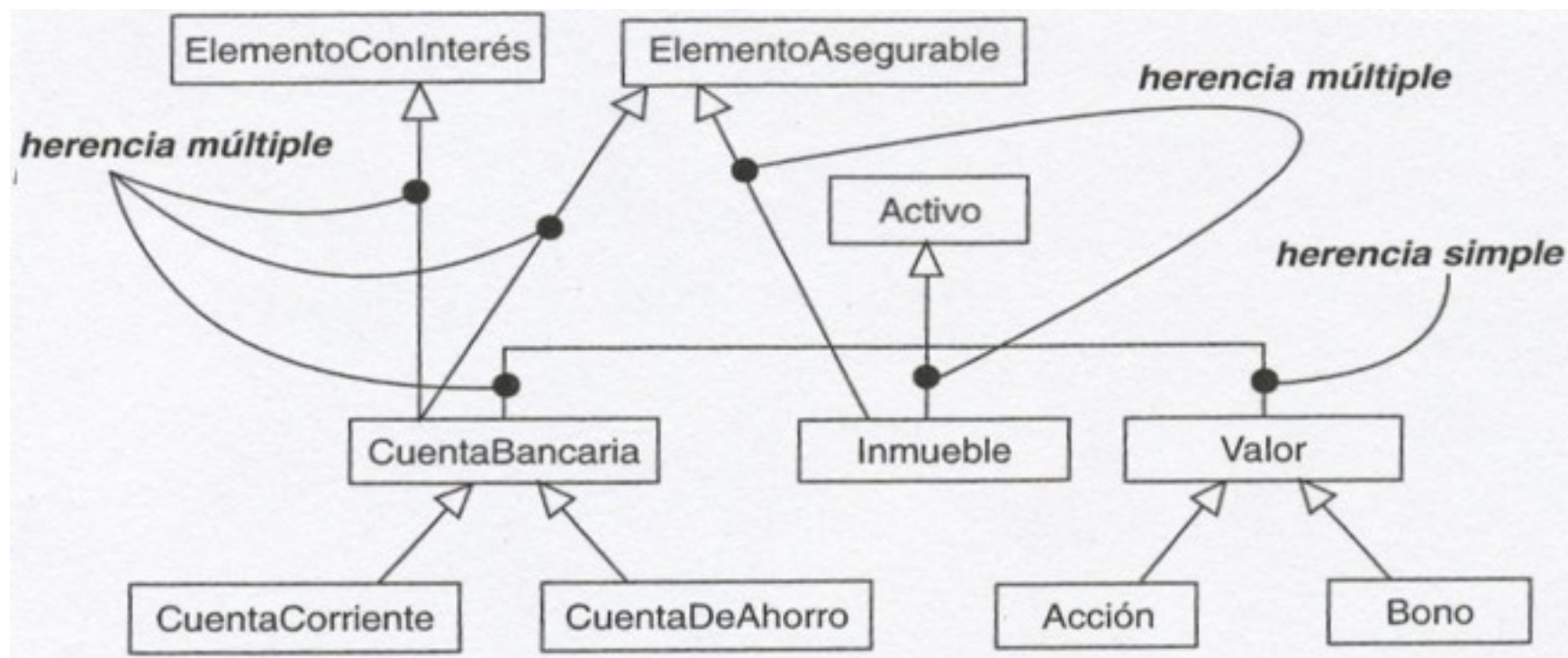
- Ejemplo de **jerarquía de generalización**:





Elementos Estructurales – Relaciones de Generalización

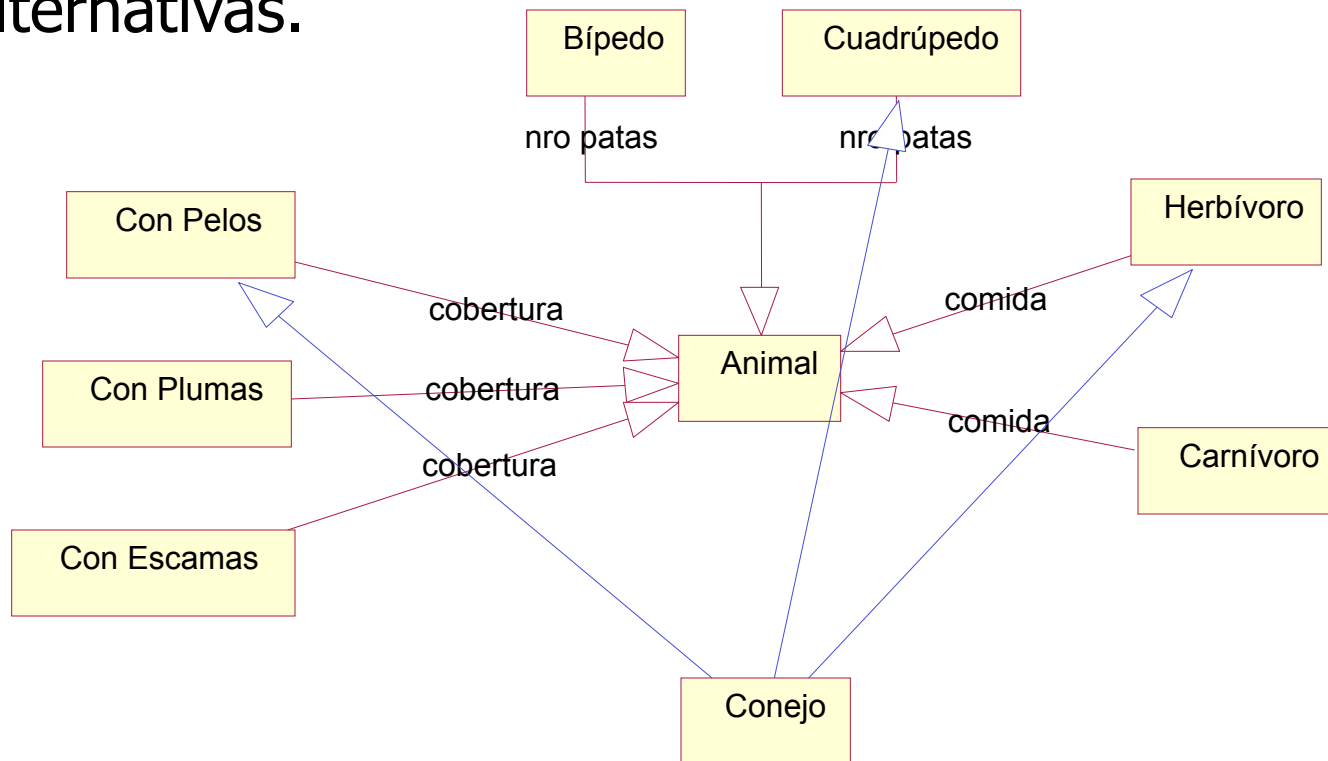
- Según el número de padres que tiene una clase se distingue entre **herencia simple y múltiple**.





Elementos Estructurales – Relaciones de Generalización

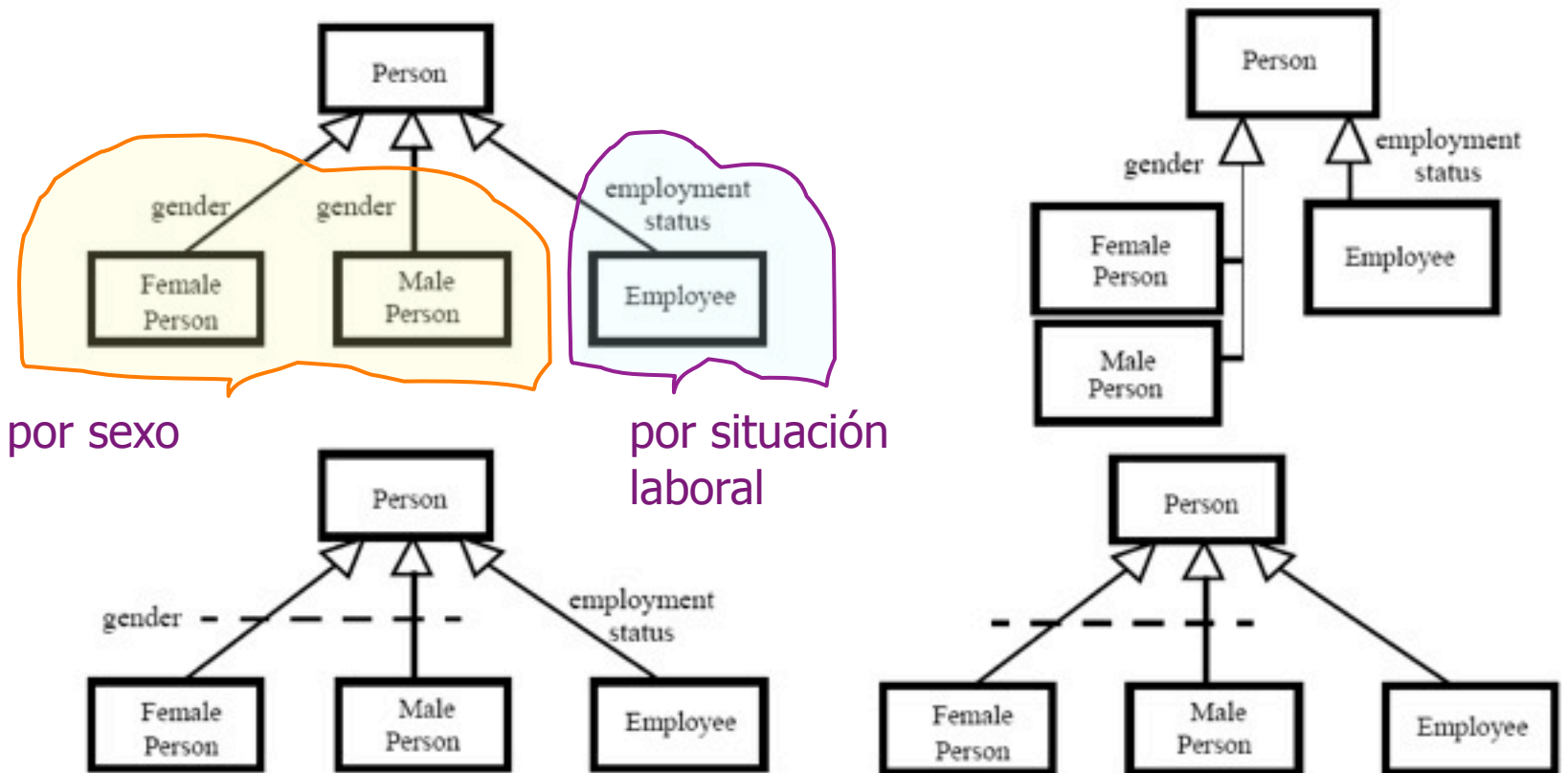
- La **herencia múltiple** puede producir **conflictos de colisiones** de nombres y de precedencia.
 - Se minimizan haciendo grupos de generalizaciones disjuntas con clases padre en hojas de jerarquías alternativas.





Elementos Estructurales – Relaciones de Generalización

- **Grupo de Generalizaciones** (generalization set)
 - Varias generalizaciones que comparten una misma superclase en base a un mismo criterio de especialización.





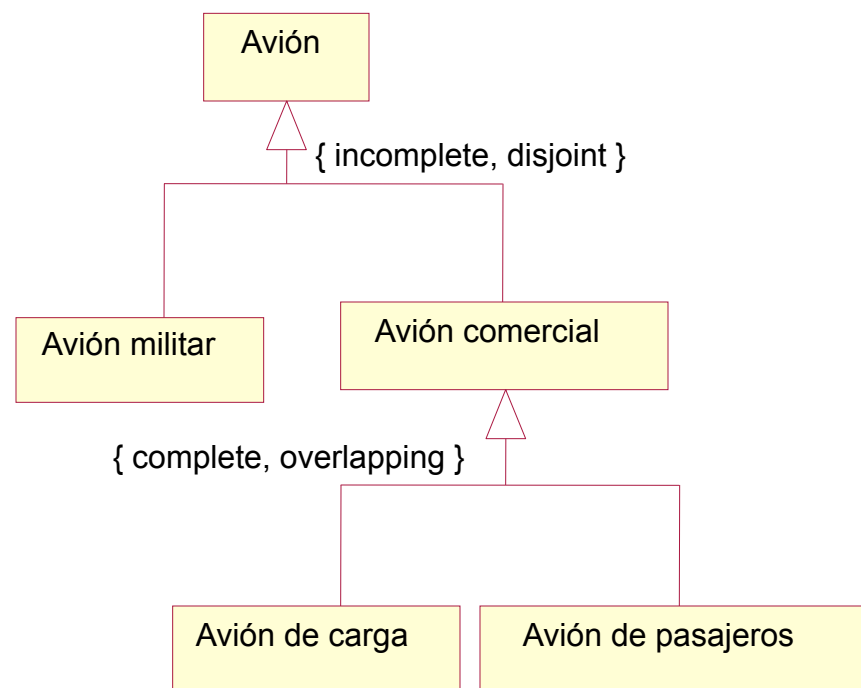
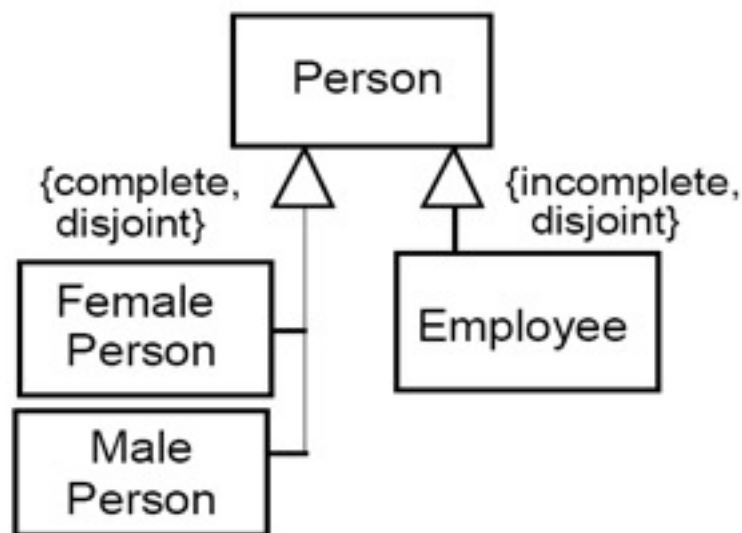
Elementos Estructurales – Relaciones de Generalización

- Los **Grupos de Generalizaciones** tienen dos **propiedades**:
 - **Cobertura** (IsCovering)
 - **complete** => cada instancia de la superclase es obligatoriamente también instancia de alguna (o varias) subclases.
 - **incomplete** => puede haber instancias de la superclase que no sean instancias en ninguna subclase.
 - **Solapamiento** (IsDisjoint)
 - **disjoint** => las subclases no pueden tener instancias comunes.
 - **overlapping** => las subclases pueden tener instancias comunes.
 - Notación: {<cobertura>, <solapamiento>}
 - Valor por defecto: {incomplete, disjoint}



Elementos Estructurales – Relaciones de Generalización

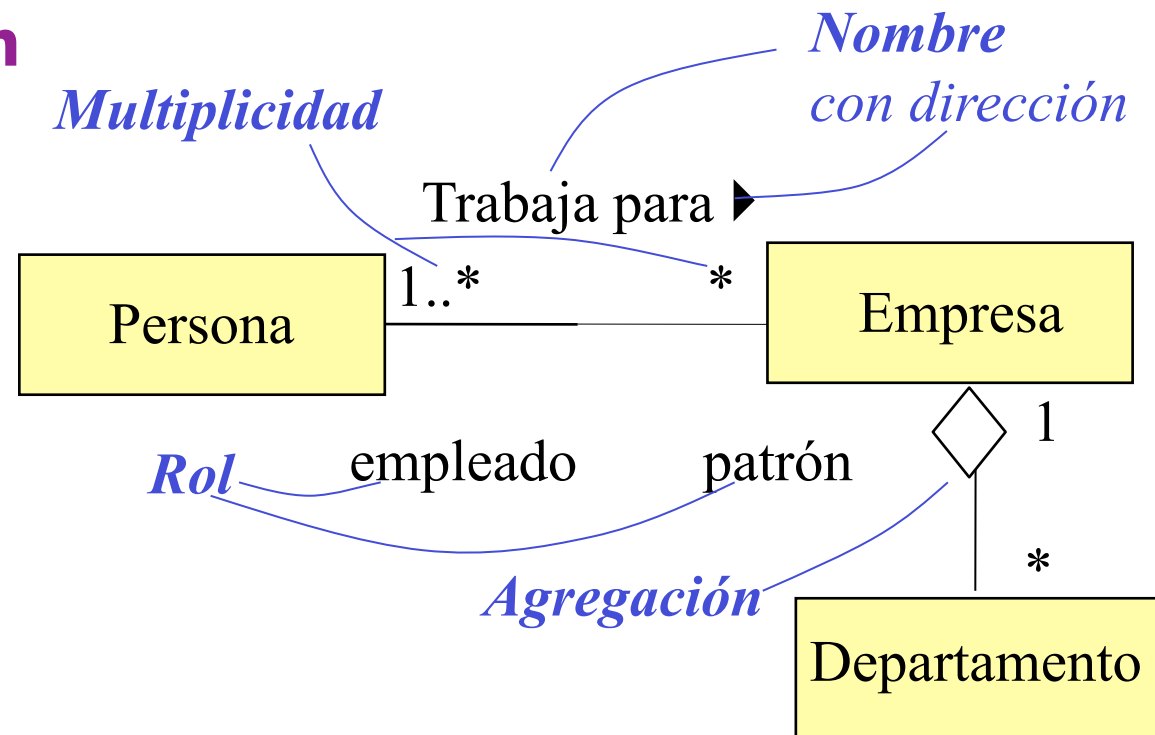
- Ejemplos de Grupos de Generalizaciones con propiedades de **cobertura y solapamiento**:





Elementos Estructurales - Relaciones de Asociación

- Existen 4 **adornos básicos** en las **asociaciones**:
 - **Nombre** (opcionalmente con **dirección**)
 - **Rol** (en cada extremo de asociación)
 - **Multiplicidad** (en cada extremo de asociación)
 - **Agregación**





Elementos Estructurales - Relaciones de Asociación

● Multiplicidad de una Asociación

- Notación similar a la multiplicidad de atributos

- Ejemplos:

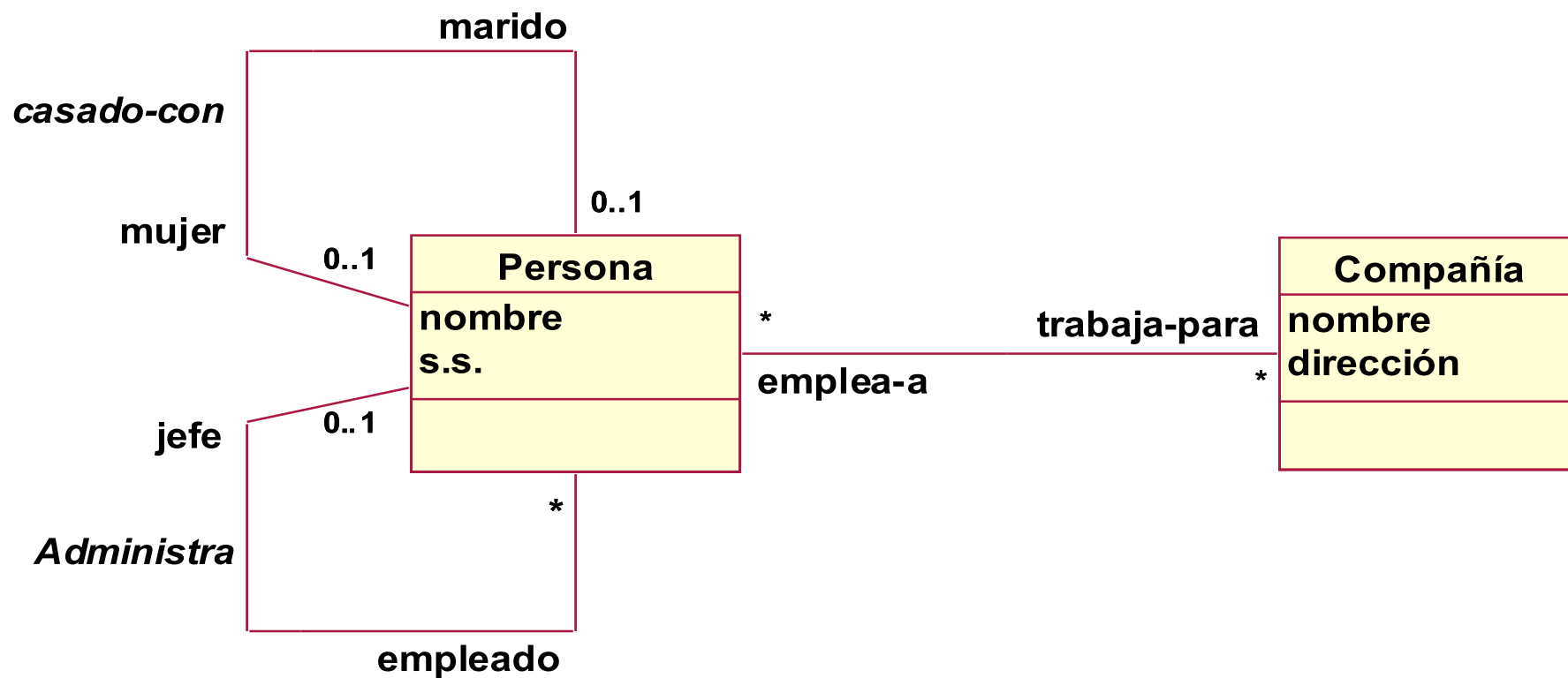
- 1 Uno y sólo uno
- 0..1 Cero o uno
- m..n Desde M hasta N (enteros naturales)
- * Cero o varios [opción por defecto]
- 0..* Cero o varios (cualquiera)
- 1..* Uno o muchos (al menos uno)

- Mínima $<>0$ \Rightarrow restricción de existencia.
- Máximo $<>*$ \Rightarrow restricción que limita el número de instancias de asociación (enlaces) por instancia de clase.



Elementos Estructurales - Relaciones de Asociación

- Ejemplos





Elementos Estructurales - Relaciones de Asociación

- Otros **adornos y características** más **avanzadas** de las **asociaciones** en UML son:
 - Navegación
 - Visibilidad
 - Calificación
 - Composición
 - Clases de Asociación
 - Modificadores



Elementos Estructurales - Relaciones de Asociación

● Navegación en Asociaciones

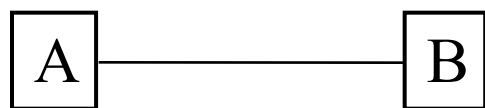
- Por defecto, una asociación en UML 2 es bidireccional =>
 - Es posible navegar desde los objetos de una clase a los objetos de la otra clase.
- Si queremos que no sea bidireccional se utiliza una flecha.
 - Esta especificación tendrá connotaciones de eficiencia en la implementación. Ejemplo:
 - Desde cada usuario se llega fácilmente a sus cuentas, pero no al contrario.



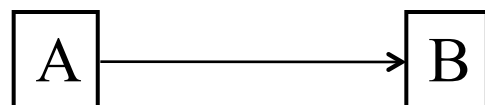


Elementos Estructurales - Relaciones de Asociación

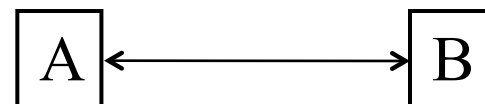
- Notación UML 2 de la Navegación en Asociaciones



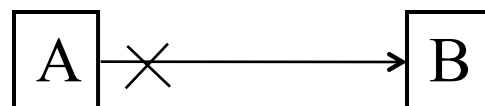
Navegabilidad indefinida



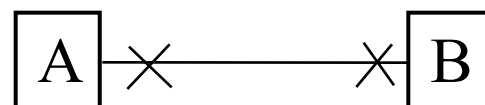
Navegable de A a B, de B a A indefinida



Navegable en ambos sentidos (no se usa)



Navegable sólo de A a B (no se usa)



No navegable en ningún sentido
(no se usa)



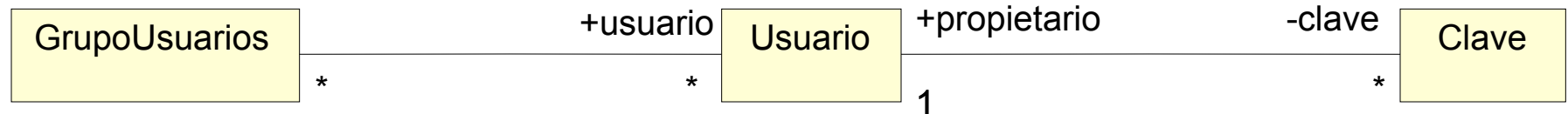
Elementos Estructurales - Relaciones de Asociación

- La **Visibilidad** en asociaciones permite controlar si los objetos de una clase pueden ver los de la otra.
- En UML 2 a cada rol (nombre de extremo) de asociación se le pueden asignar tres tipos de visibilidad:
 - **Publica (+)**: opción por defecto, sin restricciones.
 - **Privada (-)**: Los objetos de ese extremo no son accesibles por objetos externos.
 - **Protegida (#)**: Los objetos de ese extremo no son accesibles por objetos externos a la asociación, excepto los hijos del otro extremo.



Elementos Estructurales - Relaciones de Asociación

- **Ejemplo** de **Visibilidad en Asociaciones**



- Dado un usuario es posible "ver" sus objetos clave.
- Una clave es privada a un Usuario y no es visible desde el exterior de la asociación.
 - =>
 - Dado un objeto GrupoUsuarios se pueden ver sus objetos Usuario (y viceversa), pero no se pueden ver los objetos Clave de dichos Usuarios.



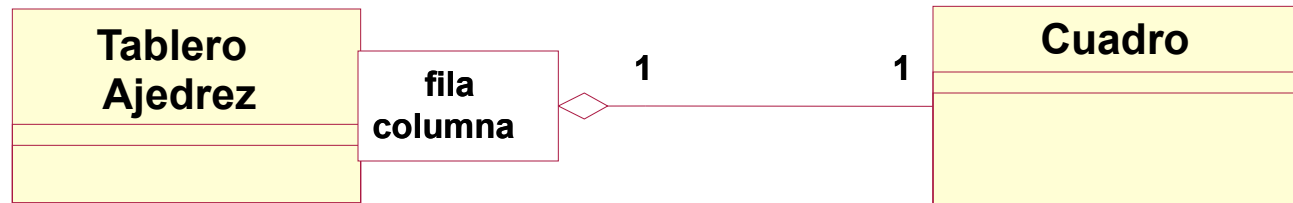
Elementos Estructurales - Relaciones de Asociación

- Un problema relacionado con las **asociaciones** es la **búsqueda**:
 - Dado un objeto de un extremo, ¿Cómo identificar un objeto o conjunto de objetos en el otro extremo?
- Para ello se emplea un **Calificador**:
 - Atributo de una asociación cuyos valores identifican un subconjunto de objetos (suele ser uno solo) relacionados con un objeto a través de una asociación.
 - El objeto origen, junto a los valores de los atributos del calificador, devuelven un objeto destino o un conjunto de objetos (dependiendo de la multiplicidad máxima del destino).
 - Notación: pequeño rectángulo junto al extremo de la asociación, con los atributos calificadores dentro.



Elementos Estructurales - Relaciones de Asociación

- Al considerar el valor de un **Calificador** se reduce la **multiplicidad** del rol opuesto.

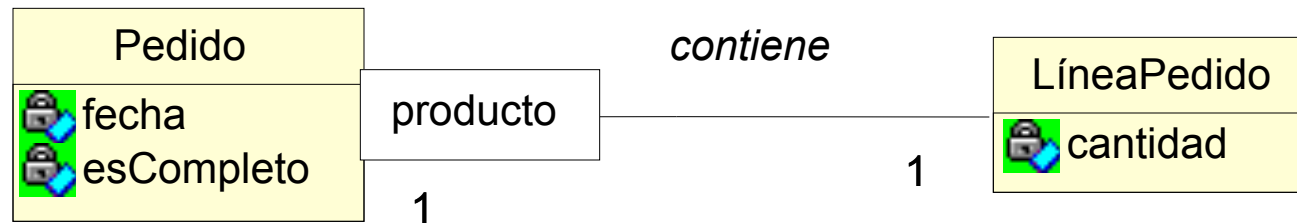


- Un Tablero de Ajedrez es una agregación de múltiples Cuadros.
- Pero en una Fila y Columna solo hay un Cuadro.



Elementos Estructurales - Relaciones de Asociación

- Niveles de significado de las **Calificaciones de Asociaciones**

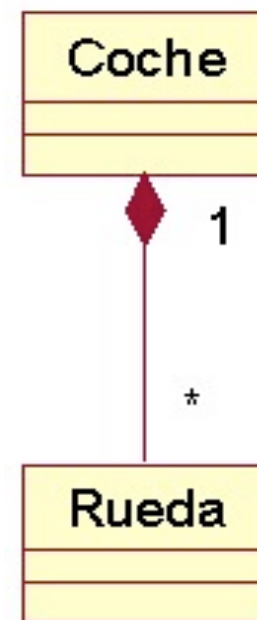


- **Conceptual:** Dentro del mismo pedido no pueden existir dos líneas con el mismo producto.
- **Análisis:** El acceso a LíneaPedido es indexado por producto.
- **Implementación:** Se usa una tabla para almacenar las líneas de pedido.



Elementos Estructurales - Relaciones de Asociación

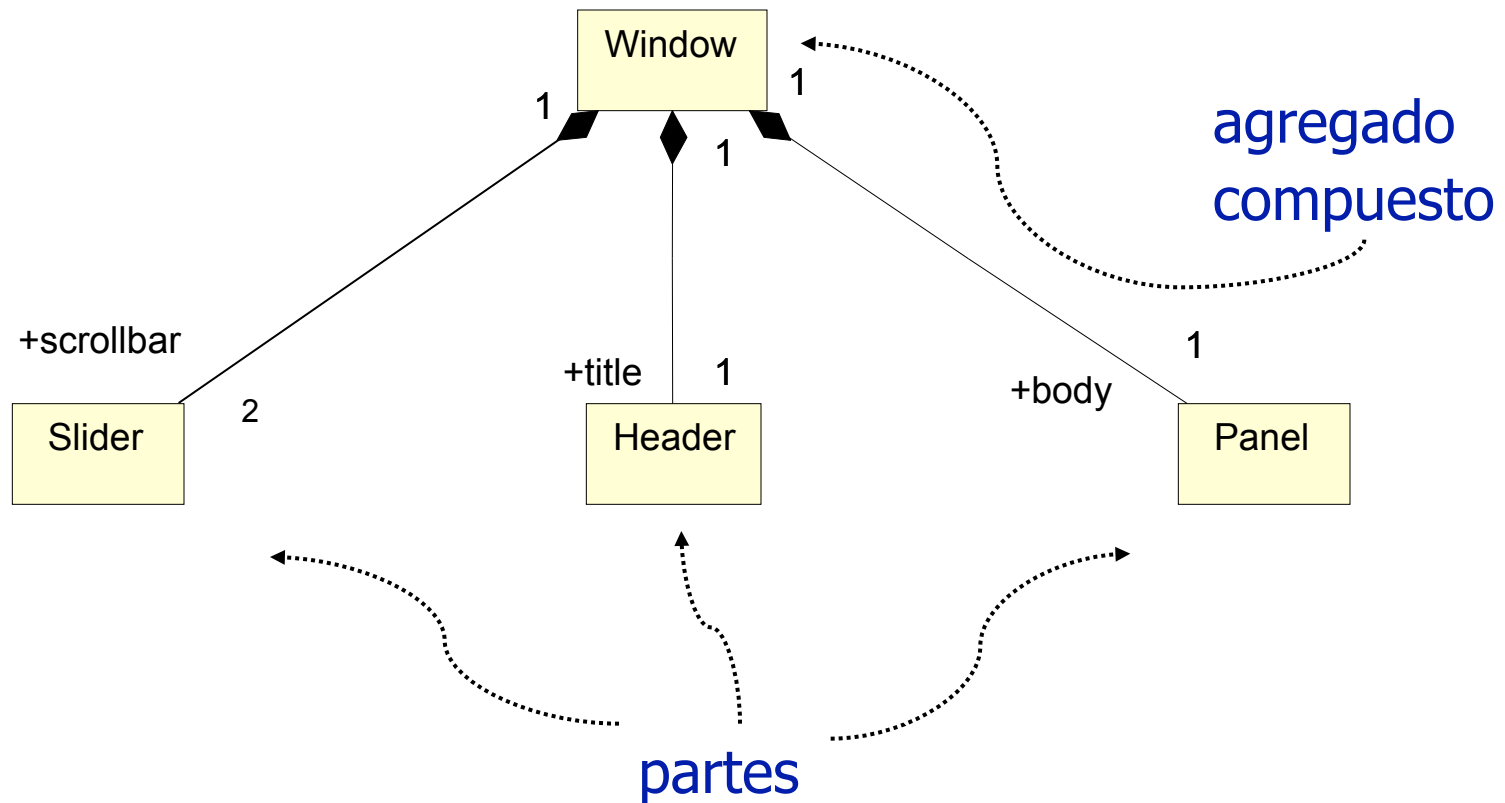
- Una **Composición** es una forma especial de **agregación** (todo-partes), pero con una fuerte relación de pertenencia y vidas coincidentes entre las "partes" y el "todo" (compuesto).
 - Una parte pertenece a un único agregado (exclusividad).
 - Si se elimina un agregado se eliminan todas sus partes (dependencia existencial).
 - Una parte se puede añadir o eliminar en cualquier instante al agregado.





Elementos Estructurales - Relaciones de Asociación

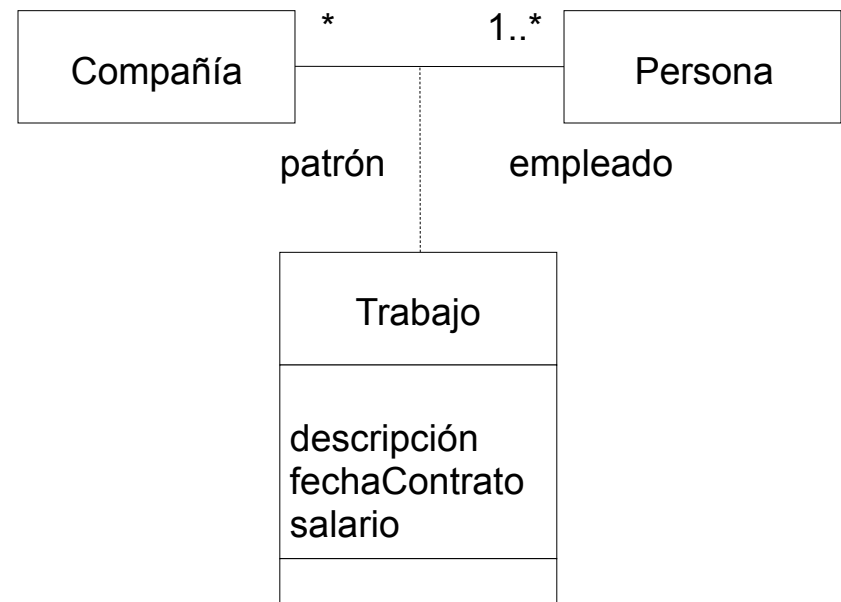
- **Ejemplo** de una **Composición**





Elementos Estructurales - Relaciones de Asociación

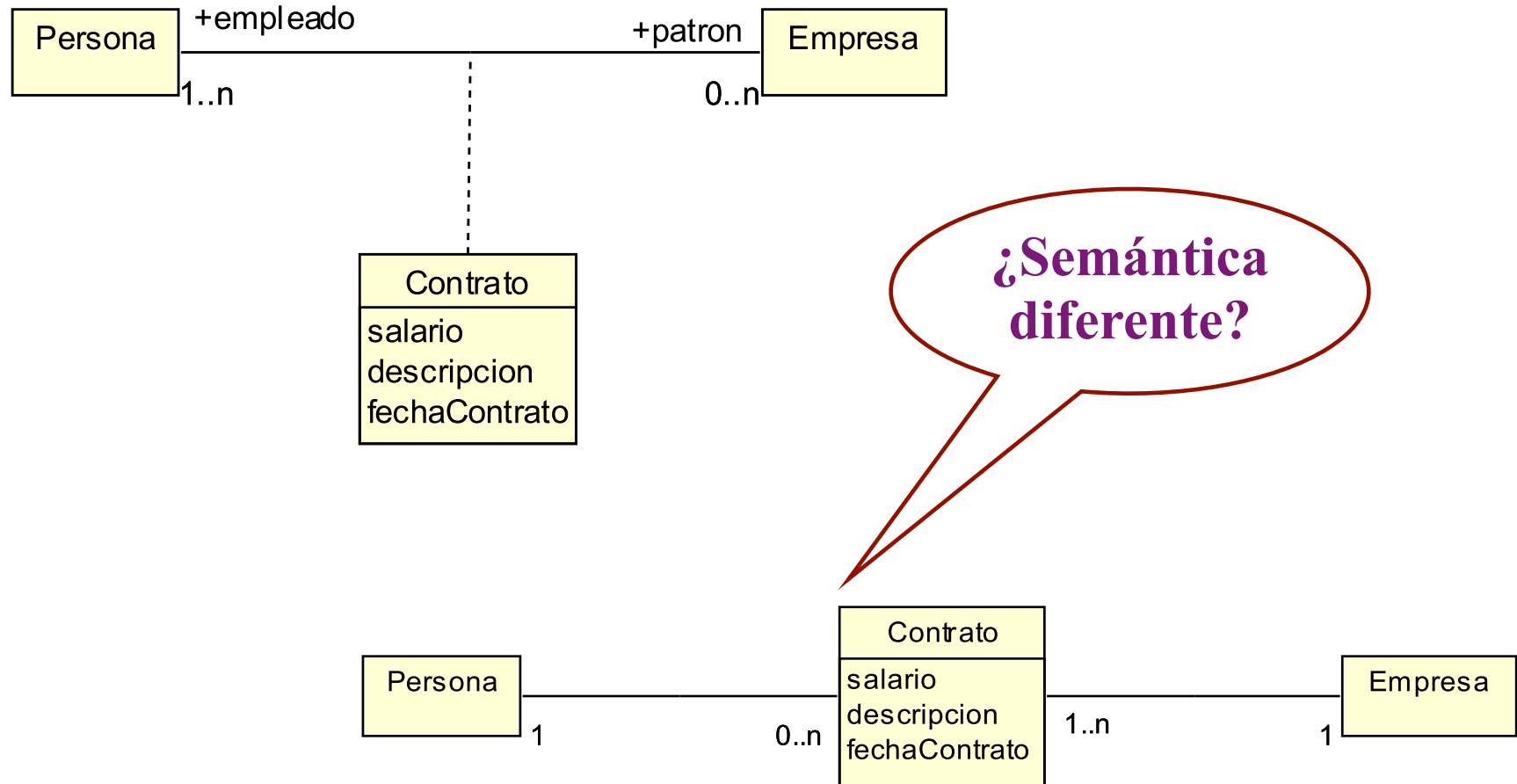
- Para representar las propiedades (atributos) de una asociación se emplea una **Clase Asociación**.
 - Añade una **restricción**:
"Sólo puede existir una instancia de la asociación entre cualquier par de objetos participantes".
 - En el ejemplo no se permite que una persona tenga varios trabajos para la misma compañía.





Elementos Estructurales - Relaciones de Asociación

- **Ejemplo** de una **Clase Asociación**





Elementos Estructurales - Relaciones de Asociación

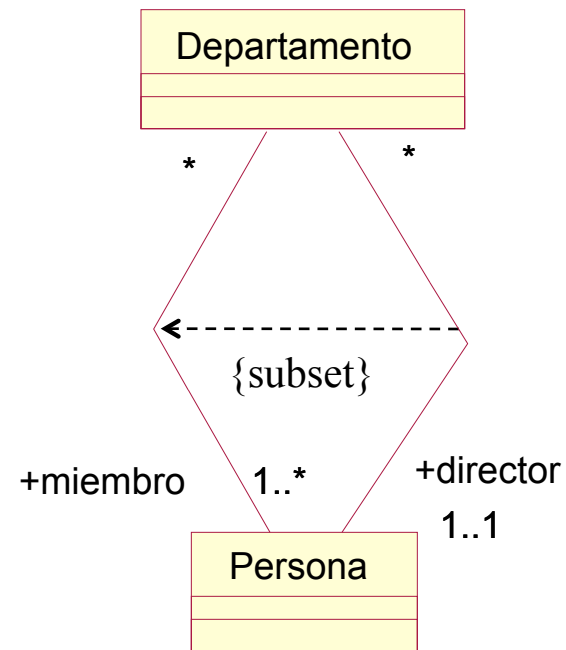
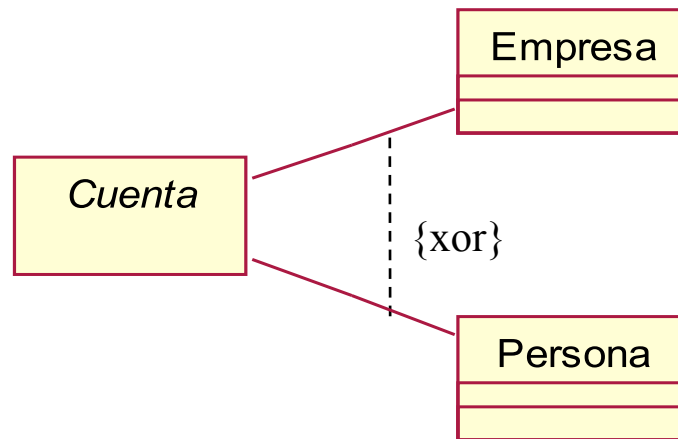
- A los **extremos de asociación** se les pueden aplicar **modificadores**:
 - **Orden (ordered)**: El conjunto de objetos de un extremo de una asociación (con multiplicidad mayor que uno) están ordenados.
 - **Unicidad**: Los objetos del extremo de asociación son únicos o no. Combinado con la anterior se pueden establecer cuatro situaciones:
 - **set** - objetos únicos, sin duplicados.
 - **bag** - objetos no únicos, puede haber duplicados.
 - **ordered set** - únicos ordenados.
 - **sequence** - ordenados con duplicados.
 - **Cambiabilidad (readonly)**: Una vez añadido un enlace desde un objeto del otro extremo, no se puede modificar ni eliminar.





Elementos Estructurales - Restricciones entre Relaciones

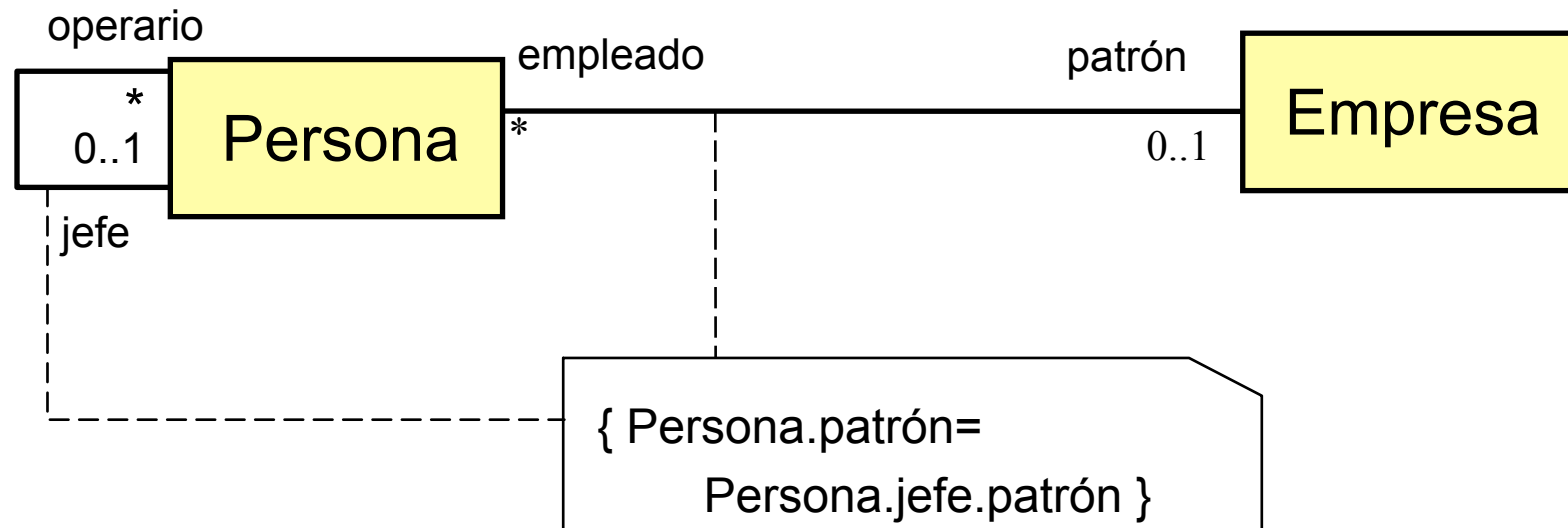
- Restricciones entre asociaciones
- UML 2 incluye algunas restricciones inter-relación predefinidas de uso habitual.





Elementos Estructurales - Restricciones entre Relaciones

- El resto de **restricciones inter-relación** se pueden definir de la manera ya conocida.
 - Ejemplo: “Una persona trabaja para la misma empresa que su jefe”



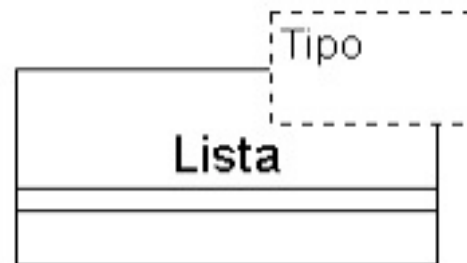


Elementos Estructurales – Clases Especiales

- **Clases Especiales**

- **Clase Plantilla** (Template)

- Es un elemento de UML que permite definir una **familia de clases**.
- Incluye “huecos” (**parámetros**) para clases, objetos y valores.
 - Se representan dentro de un rectángulo discontinuo en la esquina superior derecha del símbolo de la clase.

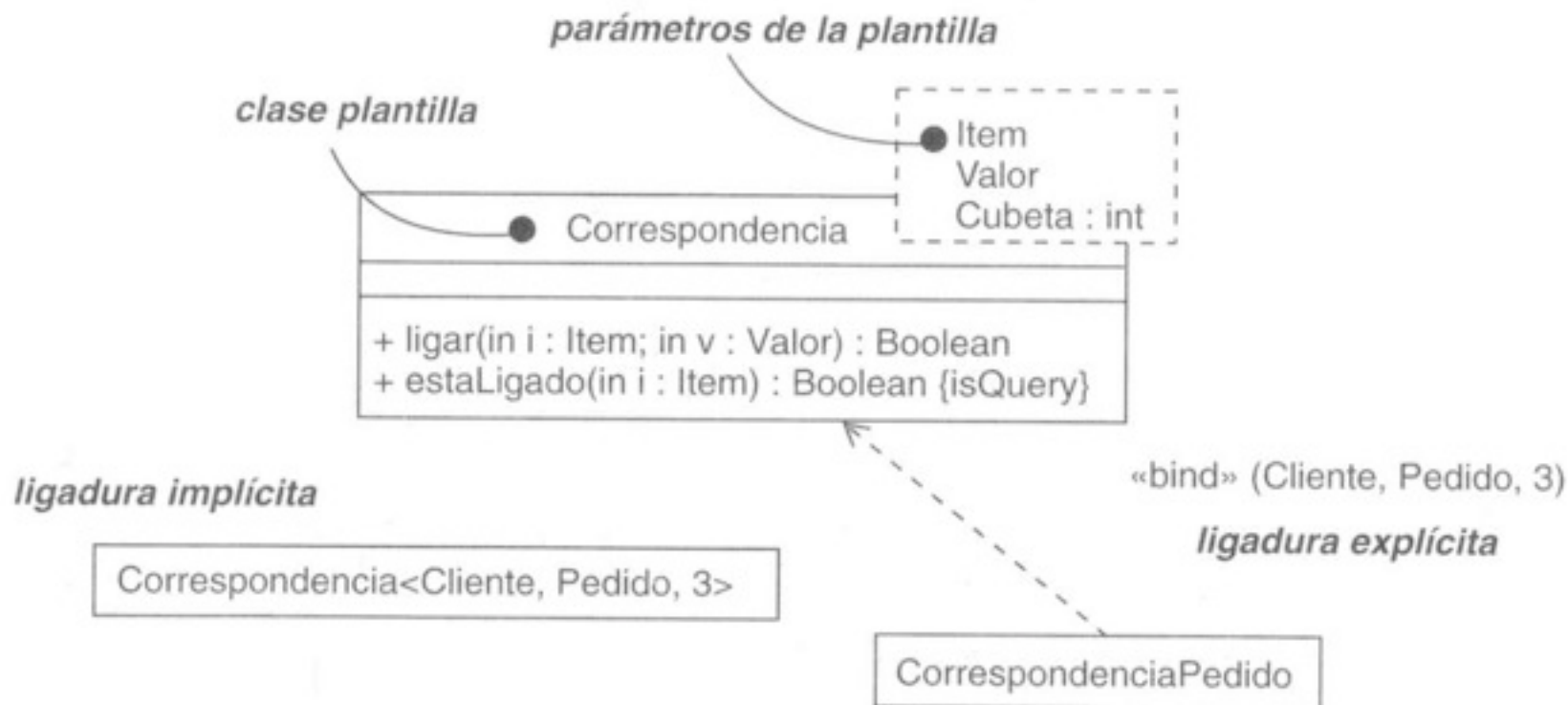


- No se puede utilizar directamente ya que debe ser instanciada antes.



Elementos Estructurales – Clases Especiales

- **Ejemplo de Clase Plantilla**



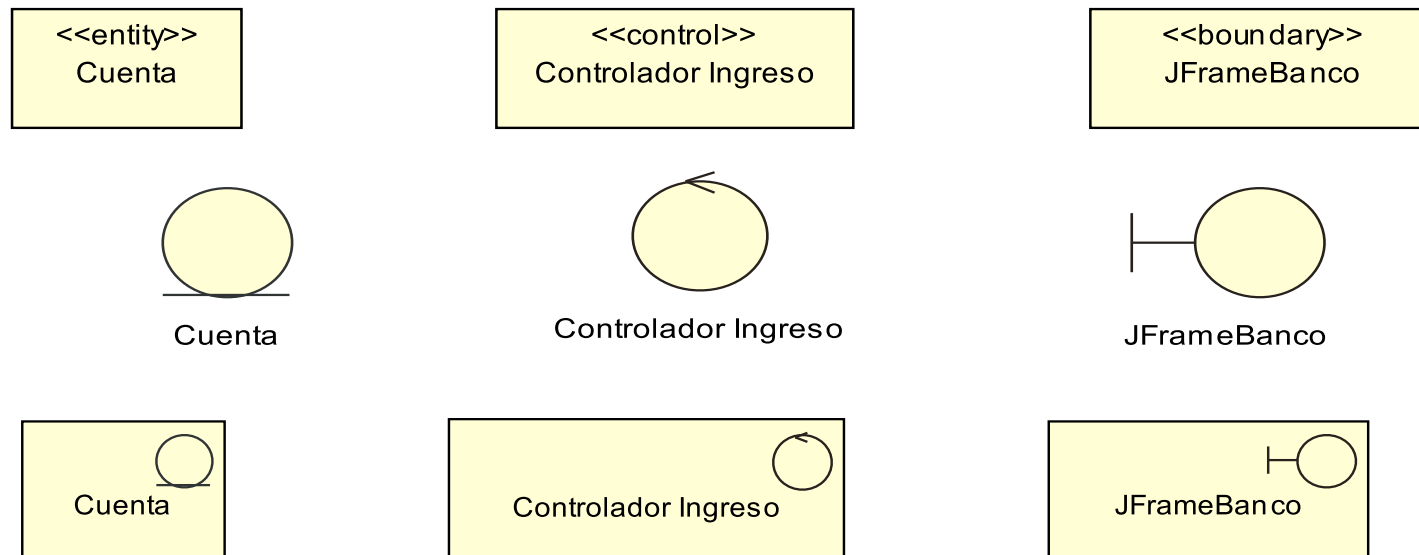
- Las clases plantillas se pueden instanciar mediante ligaduras implícitas (declarando una clase con el nombre de la plantilla) o explícitas (usando la dependencia estereotipada `<<bind>>`).



Elementos Estructurales – Clases Especiales

● Clases Estereotipadas

- UML 2 incluye varios estereotipos predefinidos de clase.



- Estereotipos de las tres categorías de clases de análisis (entidad, control e interfaz).



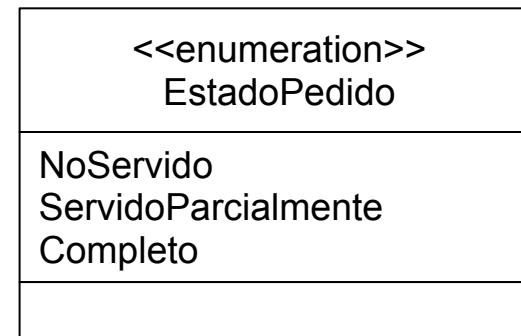
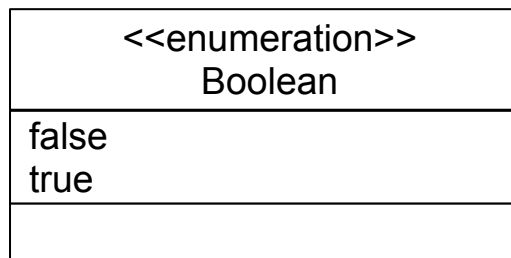
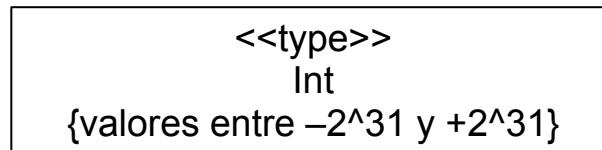
Elementos Estructurales – Clases Especiales

- Otros **estereotipos predefinidos** aplicables a las **clases** son:
 - **metaclass**: sus objetos son clases.
 - **powertype**: sus objetos son clases hijas de una clase padre específica.
 - **stereotype**: el clasificador es un estereotipo aplicable a otros elementos.
 - **utility**: clase cuyos atributos y operaciones son estáticos.
 - **auxiliary**: da soporte a otra clase mas importante.
 - **focus**: define la lógica central para una o más clases auxiliares.



Elementos Estructurales – Clases Especiales

- Las clases sirven también para especificar **tipos de datos**.
 - Se utilizan los **estereotipos "type"** (para basarse en un tipo predefinido) y **"enumeration"** para un tipo definido por enumeración.
 - Si se necesita especificar el rango de valores, hay que utilizar restricciones.





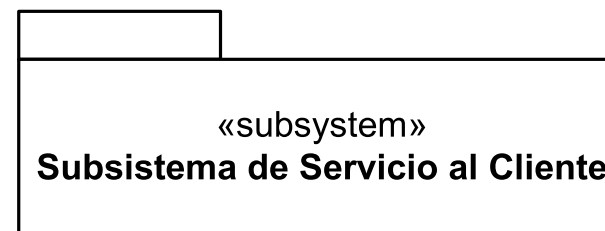
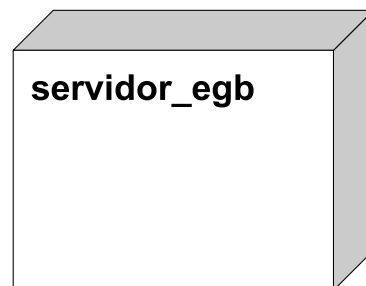
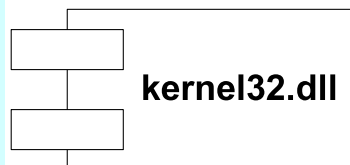
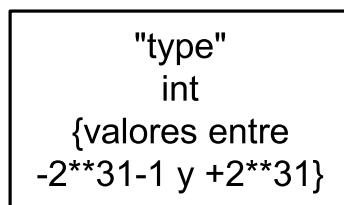
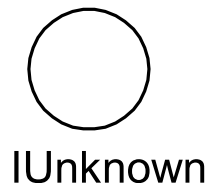
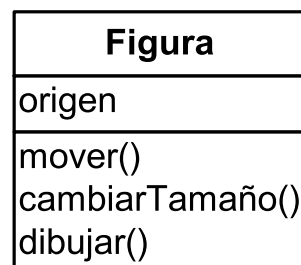
ELEMENTOS ESTRUCTURALES – OTROS CLASIFICADORES

- Los elementos de modelado que pueden tener instancias se llaman **clasificadores**.
- Aunque la clase es el clasificador más importante, en UML 2 existen otros:
 - **Interfaz**: Colección de operaciones que especifican un servicio de una clase o componente.
 - **Tipo de datos**: Tipo cuyos valores no tienen identidad, incluyendo los tipos primitivos definidos (números y strings), así como los tipos enumerados (booleanos, etc.).
 - **Señal**: Especificación de un estímulo asíncrono enviado entre instancias.
 - **Componente**: Parte física y reemplazable de un sistema que es conforme a y proporciona la realización de un conjunto de interfaces.
 - **Nodo**: Elemento físico que existe en tiempo de ejecución y representa un recurso computacional (generalmente una máquina).
 - **Caso de uso**: Descripción de una secuencia de acciones, incluyendo variantes, que ejecuta un sistema y produce un resultado observable para un actor particular.
 - **Subsistema**: Agrupación de elementos, algunos de los cuales constituyen una especificación del comportamiento de los otros elementos contenidos.



ELEMENTOS ESTRUCTURALES – OTROS CLASIFICADORES

- **Iconos** de algunos **clasificadores**.





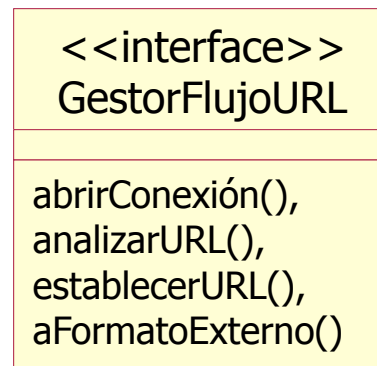
3. Interfaces

- Definen una línea entre la **especificación** de lo que una abstracción hace y la **implementación** de cómo lo hace.
- Una **interfaz** es una **colección de operaciones** que especifican los servicios de una clase o componente.
 - Se utilizan para modelar las líneas de separación dentro de un sistema.
- Una interfaz proporciona una **separación clara** entre las **vistas externa e interna** de una abstracción, haciendo posible comprenderla sin tener que entrar en los detalles de su implementación.



Interfaces

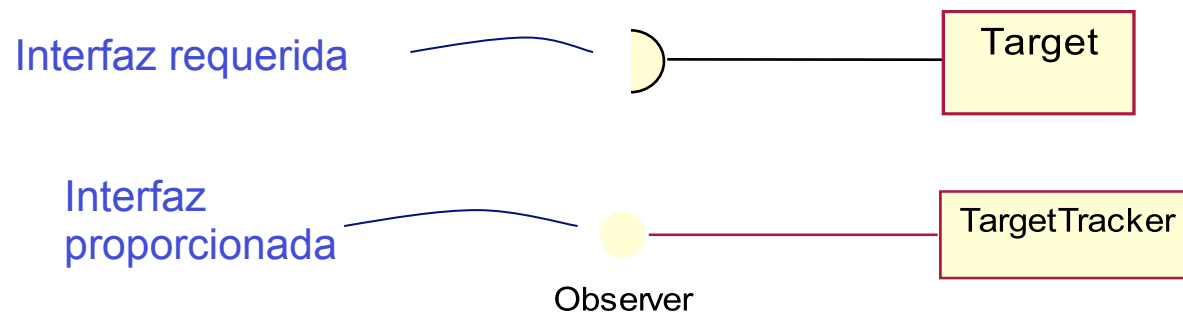
- Otros Usos:
 - En sistemas grandes se pueden emplear también para especificar la vista externa de un paquete o subsistema.
 - Especificar un contrato para un caso de uso o un subsistema.
- Se declaran mediante la clase estereotipada <<interface>>





Interfaces

- En **UML 2** existe una **notación** especial para mostrar la relación entre una clase y sus interfaces:



- **Interfaz proporcionada:** representa servicios prestados por la clase.
- **Interfaz requerida:** servicios que una clase espera de otra.



Interfaces

- Características de las Interfaces:

- Nombre.

- Simple
- Con camino: `<paquete>::<interfaz>`
 - Ejemplo: `Sensores::IDestino`

- Operaciones.

- Se pueden adornar con las técnicas ya vistas.

- Relaciones.

- Puede intervenir en relaciones de generalización, asociación y dependencia como lo hacen las clases.
- Además intervienen en relaciones de realización.

- No especifican estructura

- => No tienen **Atributos**.



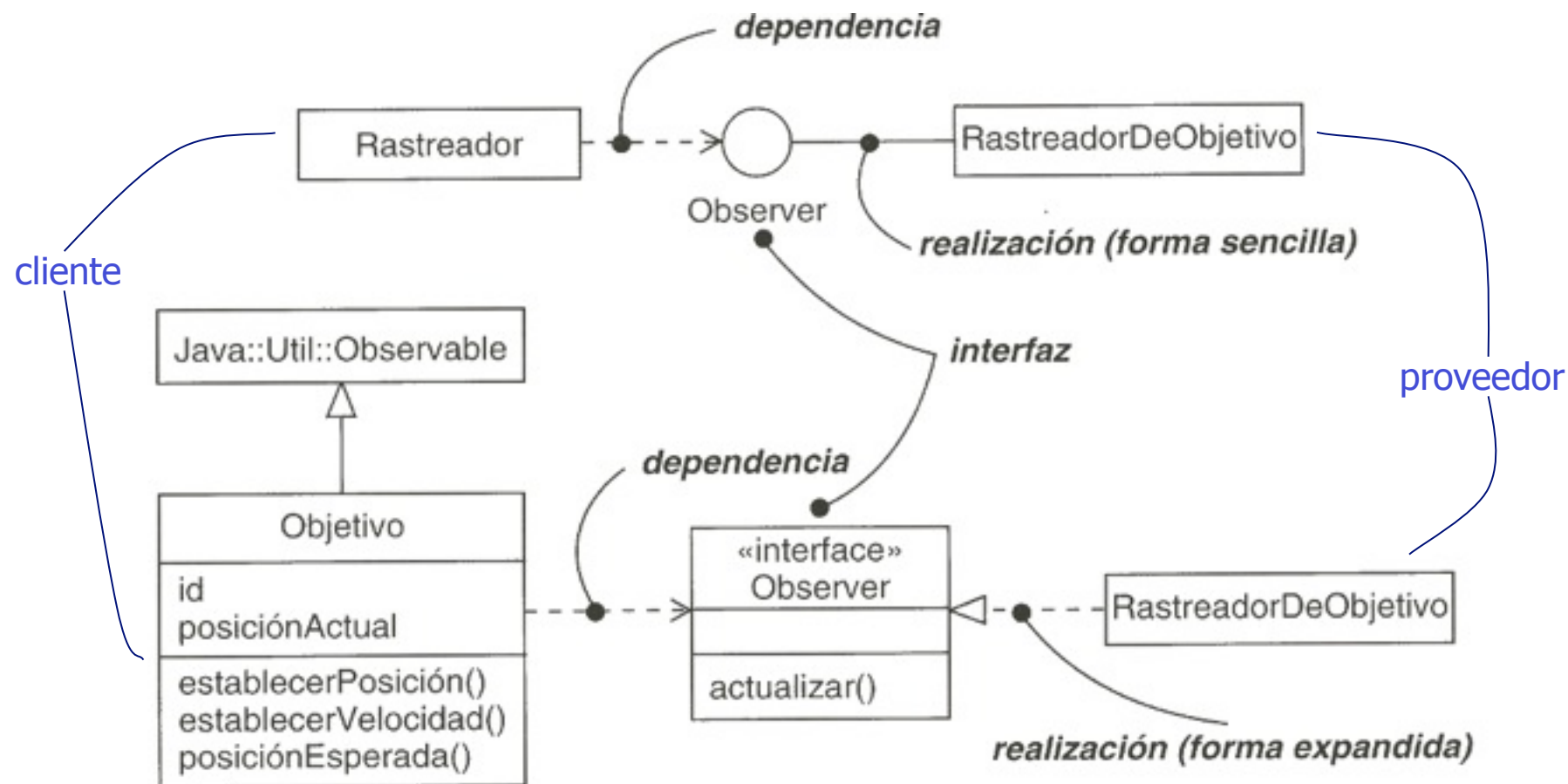
Interfaces

- Para **ampliar la especificación de una interfaz** se puede ampliar con:
 - Pre y post-condiciones
 - Los clientes que necesitan usar la interfaz será capaz de entender qué hace y cómo utilizarla, sin necesidad de indagar su implementación.
 - Una máquina de estados
 - Para especificar el orden parcial permitido de las operaciones de la interfaz.
 - Colaboraciones
 - Para especificar el comportamiento esperado a través de una serie de diagramas de interacción.



Interfaces - Realización

- Las **realizaciones de interfaces** se pueden representar de dos formas:





Diagramas de Clases

4. Diagramas de Clases

- Tienen múltiples utilidades:
 - Modelar la vista de diseño estática de un sistema (para soportar los requisitos funcionales):
 - Vocabulario del sistema
 - Colaboraciones, y
 - Esquemas de datos.
 - Son la base para los diagramas de componentes y los de despliegue.
 - Para construir sistemas ejecutables aplicando ingeniería directa e inversa.



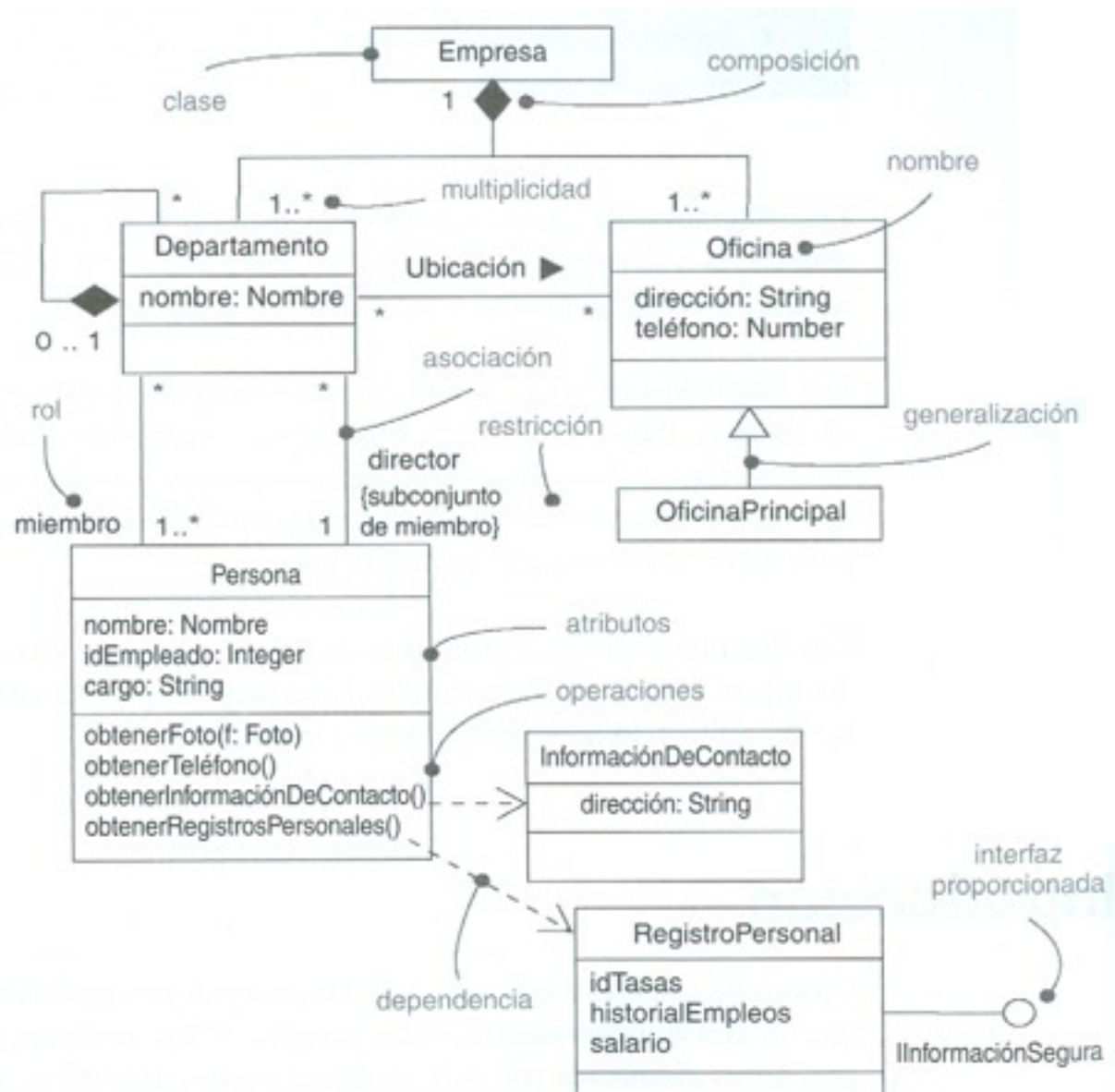
Diagramas de Clases

- **Contenido** de un Diagrama de Clases
 - Clases
 - Interfaces
 - Colaboraciones
 - Relaciones
 - Dependencia
 - Generalización
 - Asociación
 - Notas
 - Restricciones
 - Paquetes y subsistemas (agrupar elementos)
 - Instancias



Diagramas de Clases

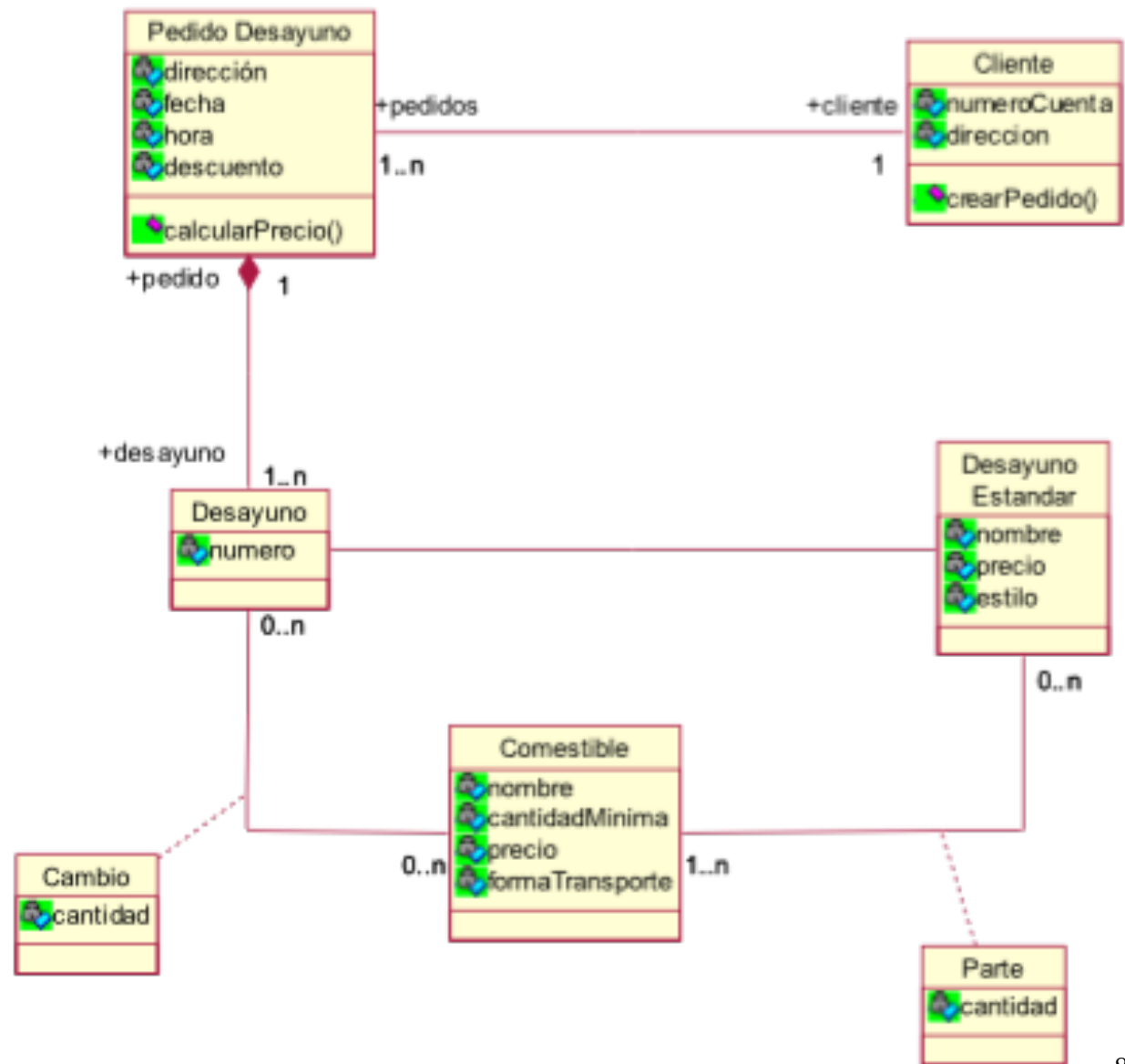
- Ejemplo de Diagrama de Clases**





Diagramas de Clases

- **Ejemplo de Diagrama de Clases**

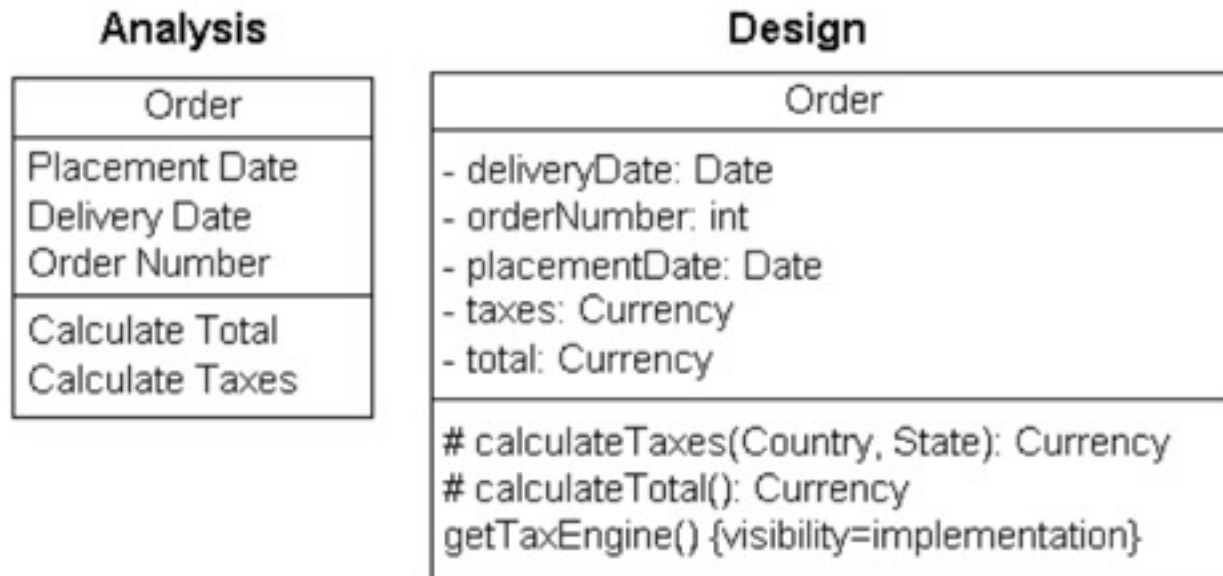




Diagramas de Clases

- **Análisis vs Diseño**

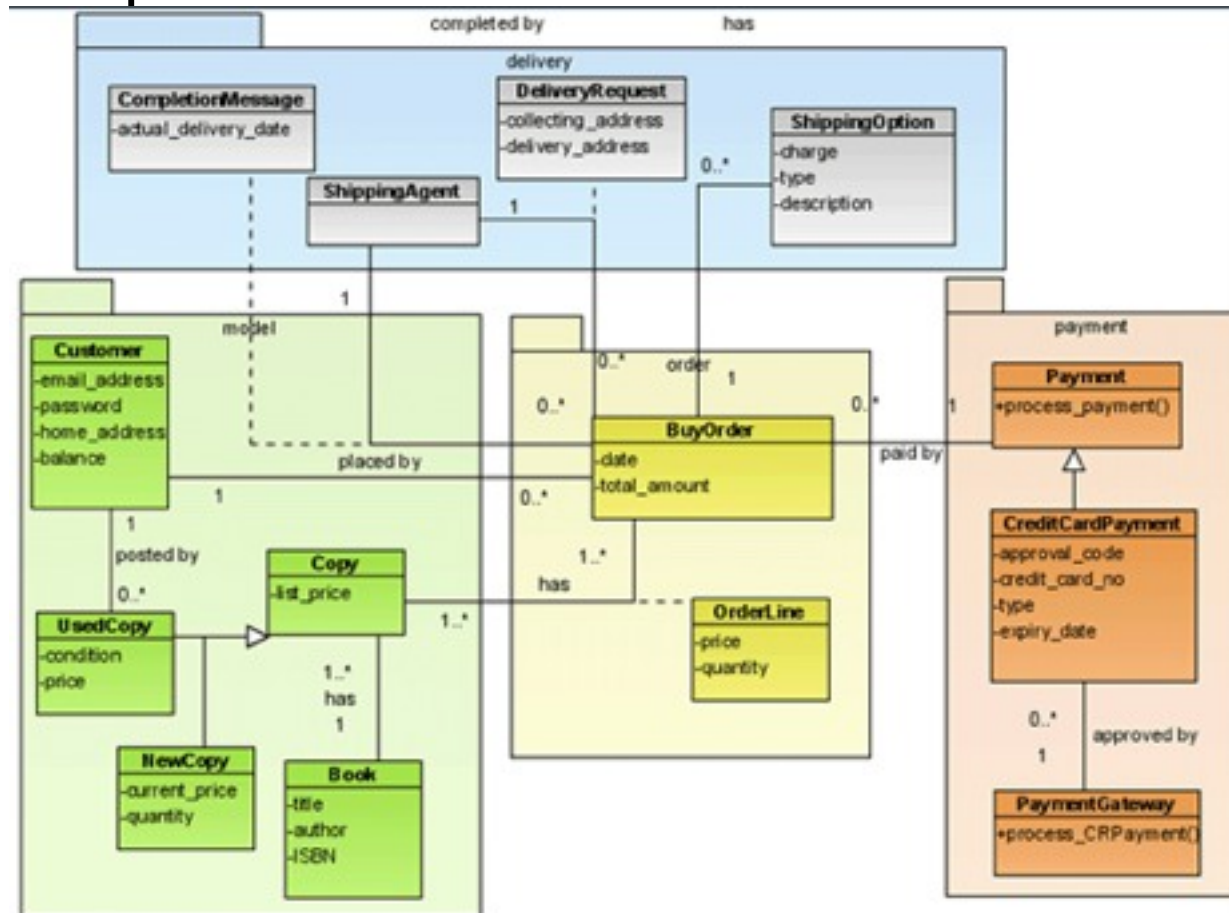
- Los adornos de las clases de diseño son más completos que en las clases de análisis.





Diagramas de Clases

- Los **diagramas de clase complejos** se pueden manejar de forma más sencilla usando **paquetes** para agrupar elementos próximos.





Diagramas de Clases - Consejos

- **Consejos**
- Al **modelar clasificadores**:
 - Elegir el tipo de clasificador (clase, interfaz, componente, etc.) que mejor se adapte a la abstracción.
- Un **clasificador** está **bien estructurado** si:
 - Tiene aspectos tanto estructurales como de comportamiento.
 - Tiene cohesión fuerte y acoplamiento débil.
 - Muestra solo lo necesario para ser usado y oculta lo demás.
 - Evita la ambigüedad en su objetivo y en su semántica.
 - Debe dar libertad a los implementadores evitando la sobre-especificación.
 - Al contrario, no debe tener ambigüedad en significado por estar infra-especificado.



Diagramas de Clases - Consejos

- Al **modelar clases**:
 - Cada clase debe corresponderse con una abstracción tangible o conceptual en el dominio del usuario final o del implementador.
- Una **clase** está **bien estructurada** si:
 - Ofrece una abstracción precisa.
 - Contiene un conjunto pequeño bien definido de responsabilidades.
 - Proporciona una distinción clara entre la especificación de la abstracción y su implementación.
 - Es comprensible y sencilla, a la vez que extensible y adaptable.



Diagramas de Clases - Consejos

- Al **dibujar un clasificador**:
 - Mostrar sólo aquellas propiedades importantes para comprender la abstracción en su contexto.
 - Elegir una versión con estereotipo de forma que proporcione la mejor imagen visual de su propósito.
- Si se trata de **dibujar una clase**:
 - Organizar las listas largas de atributos y operaciones, agrupándolos de acuerdo a su categoría.
 - Mostrar las clases relacionadas en el mismo diagrama.



Diagramas de Clases - Consejos

- Una **relación bien estructurada** se caracteriza porque:
 - Sólo muestra las características necesarias para ser usada, ocultando las demás.
 - No es ambigua en su objetivo y semántica.
 - Da cierta libertad a los implementadores evitando la sobre-especificación.
 - No tiene ambigüedad en su significado por estar infra-especificada.



Diagramas de Clases - Consejos

- Al **modelar relaciones** elegir el tipo de relación y los adornos que mejor se adaptan a la abstracción dada:
 - Usar dependencias sólo cuando la relación no sea estructural.
 - Usar generalización sólo cuando la relación significa “es-un-tipo-de”.
 - Intentar evitar la herencia múltiple (reemplazar por agregación si se puede).
 - Evitar generalizaciones cíclicas.
 - Las jerarquías de herencia no deben ser muy profundas (menos de 6 niveles) ni demasiado anchas (reducir anchura usando clases abstractas).
 - Usar asociaciones donde existan relaciones estructurales.
 - No cuando se puede representar con parámetros o variables.



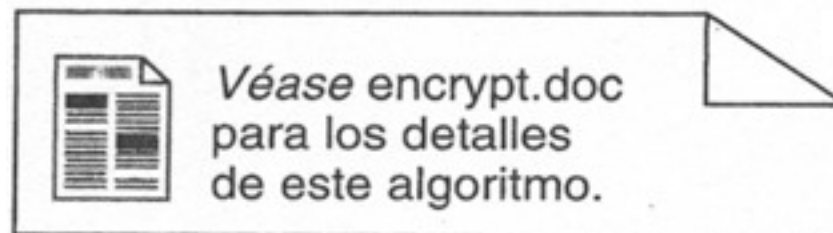
Diagramas de Clases - Consejos

- Al **dibujar relaciones**:
 - Usar líneas verticales o horizontales combinadas con oblicuas buscando
 - Aprovechar mejor el espacio en diagramas complejos.
 - Llamar la atención sobre grupos de relaciones.
 - Evitar cruces de líneas.
 - Mostrar sólo aquellas relaciones necesarias para comprender una agrupación de elementos.
 - Evitar las asociaciones redundantes.
 - Mostrar solo las propiedades de la relación que son importantes para comprenderla.
 - Elegir una versión estereotipada que muestre la mejor imagen visual de su propósito.



Diagramas de Clases - Consejos

- Al **añadir notas**:
 - Usarlas solo para los requisitos, observaciones, revisiones y explicaciones que no se pueden expresar en UML directamente.
 - Usarlas de forma similar a los post-it en papel para facilitar el seguimiento del trabajo.
- Al **dibujar notas**:
 - Los comentarios largos se deben poner en un documento aparte y usar la nota para referir a dicho comentario.





Diagramas de Clases - Consejos

- Al **extender un modelo**:
 - En cada proyecto la lista de **estereotipos, valores etiquetados y restricciones** debe estar homologada, evitando que cada ingeniero vaya por libre.
 - Elegir nombres cortos y auto-explicativos para estereotipos y valores etiquetados.
 - Indicar las restricciones en texto libre, salvo que sea necesario formalizarlas (en OCL).
- Al **dibujar dichas extensiones**:
 - Hacer un uso moderado de los estereotipos gráficos.
 - Combinar colores, sombreado e iconos buscando la sencillez.



Diagramas de Clases - Consejos

- Un **diagrama de clases bien estructurado** se caracteriza porque:
 - Se centra en comunicar **UN** aspecto de la vista de diseño estática del sistema.
 - Contiene sólo los elementos esenciales para comprender ese aspecto.
 - Ofrece detalles de forma consistente con el nivel de abstracción, mostrando sólo los adornos esenciales para su comprensión.
 - => Diferentes adornos en análisis y en diseño.
 - La semántica importante debe estar reflejada.



Diagramas de Clases - Consejos

- Al **dibujar un diagrama de clases**:
 - Su nombre debe comunicar el propósito.
 - Distribuir sus elementos para minimizar los cruces.
 - Organizar los elementos espacialmente de forma que los cercanos semánticamente también estén próximos visualmente.
 - Usar notas y colores para llamar la atención sobre aspectos importantes.
 - No mostrar demasiados tipos de relaciones. En cada diagrama suele prevalecer un tipo de relación:
 - Un diagrama para mostrar una jerarquía de generalizaciones.
 - Otro diagrama para mostrar asociaciones.



Diagramas de Clases - Consejos

- Al **modelar una interfaz**:
 - Recordar que debe representar una línea de separación en el sistema, separando especificación de implementación.
- Una **interfaz** está **bien estructurada** si:
 - Es sencilla y completa, incluyendo todas las operaciones necesarias para especificar un único servicio.
 - Es comprensible, proporcionando suficiente información para permitir su uso o su realización.
 - Es manejable, facilitando información para guiar al usuario hacia sus propiedades principales.



Diagramas de Clases - Consejos

- Al **dibujar una interfaz**:
 - Usar la **notación de piruleta** cuando solo es necesario señalar una línea de separación en el sistema.
 - Caso más frecuente en **componentes**, pero no en clases.
 - Usar la notación **expandida** cuando es necesario visualizar los detalles del servicio ofrecido.
 - Es el caso más frecuente cuando se quieren especificar las líneas de separación en un sistema asociado a un **paquete** o un **subsistema**.



Objetos

5. Objetos

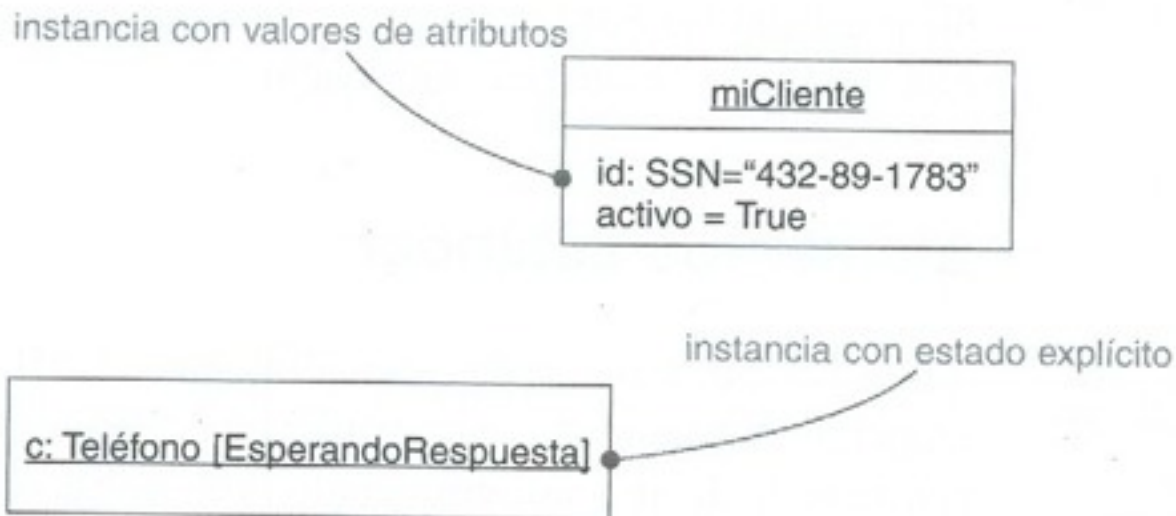
● **Objetos vs Instancias**

- En UML es común el mecanismo de abstracción de clasificación, manifestado por la dualidad Abstracción-Instancia:
 - Casos de Uso vs instancias de Casos de Uso.
 - Nodos vs instancias de Nodos.
 - Asociaciones vs instancias de asociaciones (enlaces).
- Una instancia es una manifestación concreta de una abstracción, a la que se puede aplicar operaciones y puede tener un estado (atributos).
- Los **objetos** son las **instancias de** abstracciones de tipo **clase**.



Objetos - Estado

- El **estado de un objeto** está determinado por todos los pares (<propiedad>:<valor>)
 - Incluyendo en las propiedades los atributos, enlaces (instancias de asociación) y agregaciones.



- El estado se puede identificar de forma explícita con una etiqueta entre corchetes.



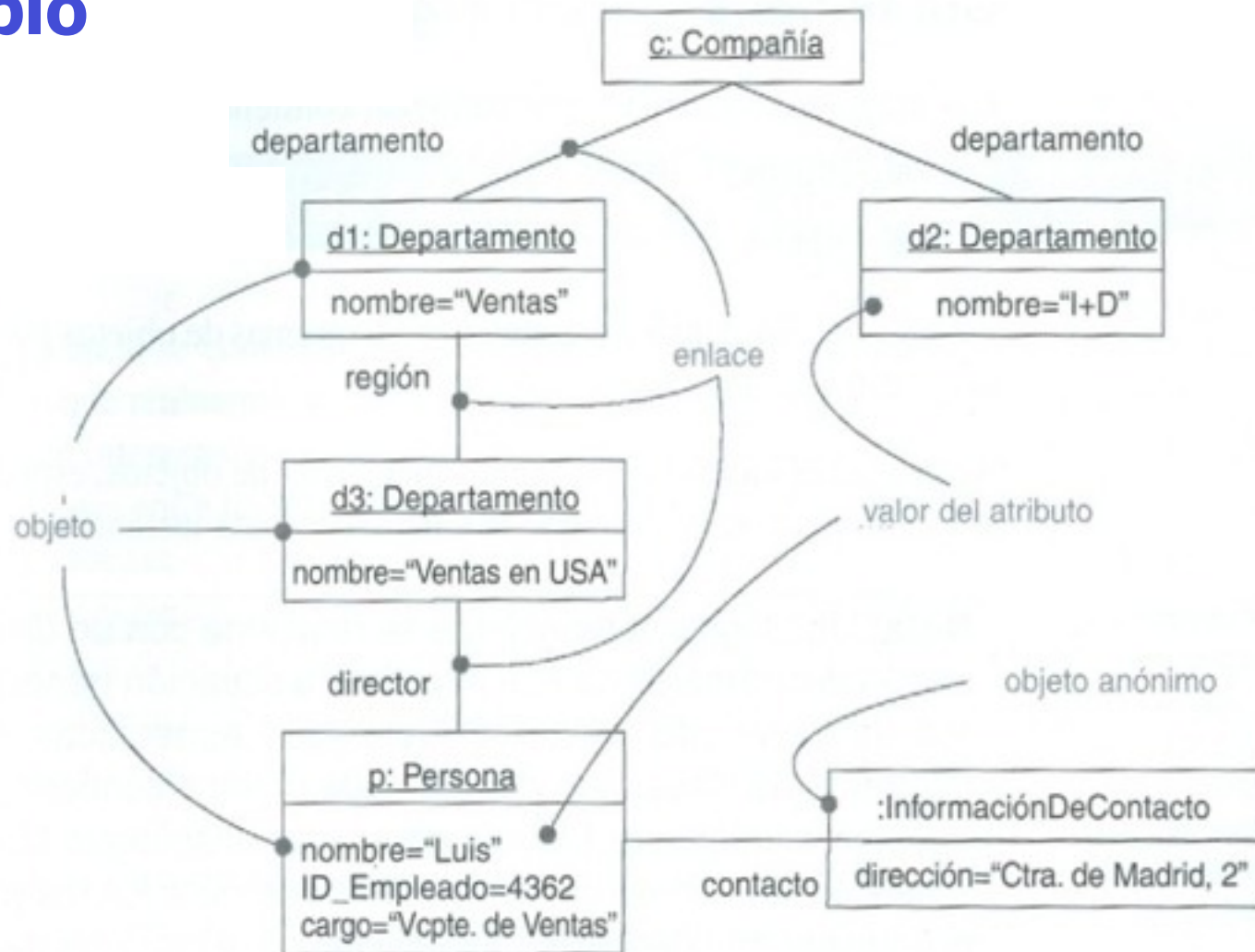
Diagramas de Objetos

- **Diagramas de Objetos**
- Sirven **para modelar**:
 - Una instancia del sistema en un momento concreto, o
 - Un conjunto de objetos, sus estados y sus relaciones;
 - mediante un conjunto de instancias de los elementos existentes en un diagrama de clases.
- **Contenido**
 - Objetos
 - Enlaces
 - Notas y restricciones
 - Clases (a veces se muestra la abstracción asociada a un objeto).



Diagramas de Objetos

- **Ejemplo**





Diagramas de Objetos - Consejos

- **Consejos**
- Al **modelar instancias**:
 - Toda instancia debe representar una manifestación de una abstracción (clase, componente, nodo, caso de uso, asociación).
 - Una **instancia** está **bien estructurada** si:
 - Está asociada explícitamente con una abstracción.
 - Tiene un nombre único extraído del vocabulario del dominio del problema o del dominio de la solución.
- Al **dibujar una instancia**:
 - Incluir el nombre de la abstracción salvo que sea obvio por el contexto.
 - Mostrar el estereotipo y el estado solo lo necesario para comprender el objeto en su contexto.
 - Las listas largas de atributos y sus valores deben agruparse por categorías.



Diagramas de Objetos - Consejos

- Un **diagrama de objetos bien estructurado**:
 - Se centra en comunicar **UN** aspecto de la vista de diseño estática o la vista de procesos estática del sistema.
 - Contiene solo aquellos elementos necesarios para comprender ese aspecto.
 - Representa una escena (un instante concreto) de la historia representada por diagrama de interacción.
 - Los detalles son consistentes con el nivel de abstracción.
 - Muestra solo los valores de atributos y enlaces necesarios para su comprensión.



Diagramas de Objetos - Consejos

- Al **dibujar un diagrama de objetos**:
 - Darle un nombre que comunique su propósito.
 - Situar los elementos minimizando los cruces de líneas.
 - Organizar espacialmente los elementos de forma que los cercanos semánticamente estén también próximos visualmente.
 - Usar notas y colores para resaltar las características importantes.
 - Incluir los valores y el estado de los objetos cuando es necesario para comunicar el propósito.



6. Modelado

● Los diagramas de clases y de objetos sirven para **modelar diversos aspectos estructurales** o estáticos de un sistema:

- Vocabulario del Sistema
- Distribución de Responsabilidades
- Semántica de una Clase
- Colaboraciones
- Esquemas de Datos
- Redes de Relaciones
- Líneas de Separación
- Instancias



Modelado - Vocabulario del Sistema

- **Vocabulario del Sistema**
- Está formado por las abstracciones que son importantes para los usuarios y para los implementadores.
 - Para **modelar el vocabulario** de un sistema:
 1. Identificar aquellas cosas (**abstracciones**) que utilizan los usuarios/ implementadores para describir el problema o la solución (tarjetas CRC, análisis con casos de uso).
 2. Identificar las **responsabilidades** de cada abstracción.
 3. Definir **atributos y operaciones** necesarios para cumplir con las responsabilidades.
- Cuando los modelos aumentan de tamaño, las abstracciones tienden a unirse en grupos relacionados conceptual y semánticamente (**paquetes**).



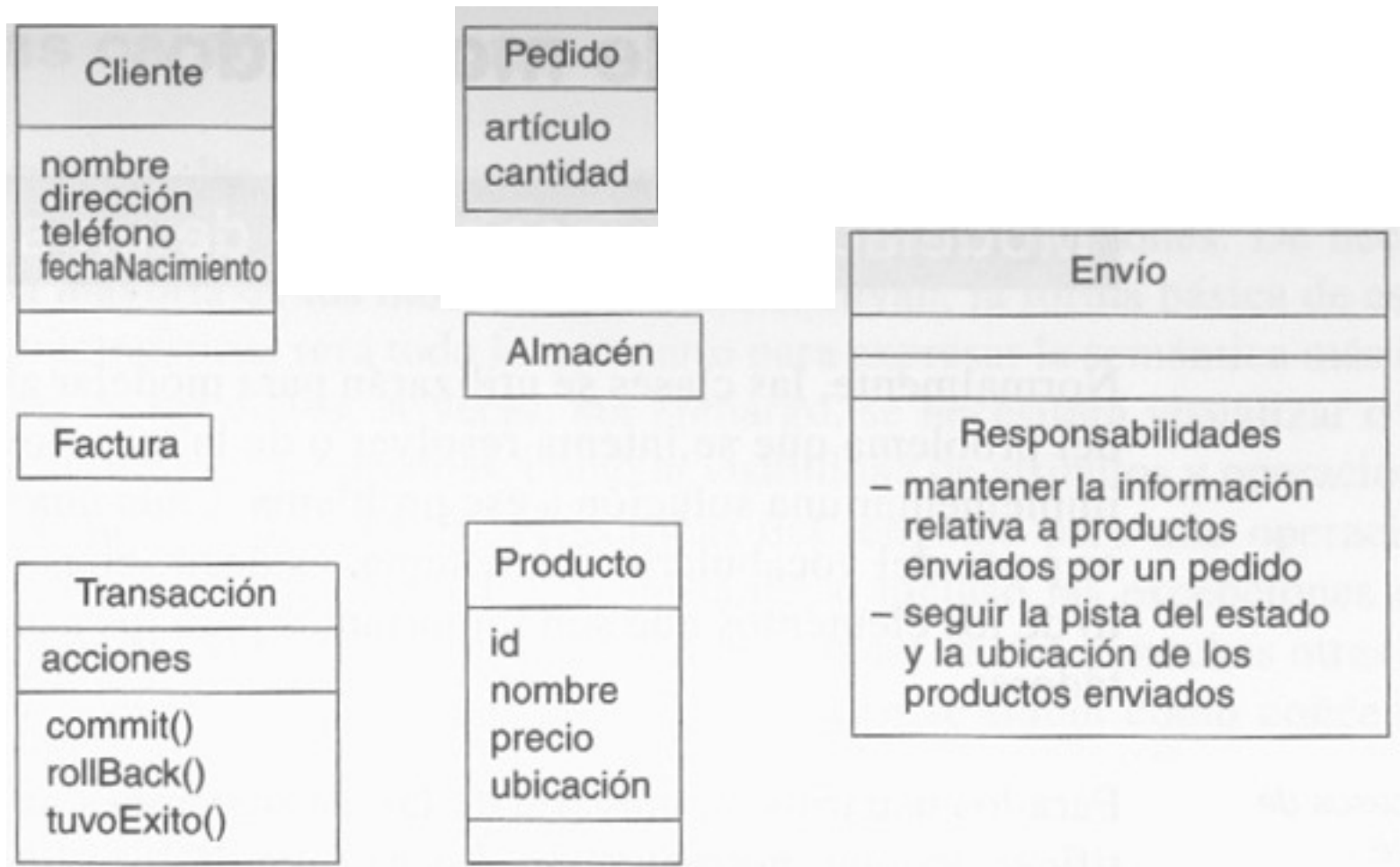
Modelado - Vocabulario del Sistema

- **Ejemplo** de un Sistema de Ventas
 - Registrar los **pedidos**, incluyendo el detalle de los **productos** y su cantidad.
 - Existe un catálogo con todos los **productos**.
 - Registrar los productos a través del código de barras o tecleando su identificación.
 - Se debe mostrar el nombre y el precio del producto.
 - Calcular el total de la **factura** ¿y los descuentos?.
 - Llevar un registro de los **clientes** y sus pedidos y facturas.
 - Actualizar automáticamente el stock de cada producto en **almacén**.
 - Llevar un control de los **envíos** asociados a cada pedido.
 -



Modelado - Vocabulario del Sistema

- **Ejemplo** de un Sistema de Ventas





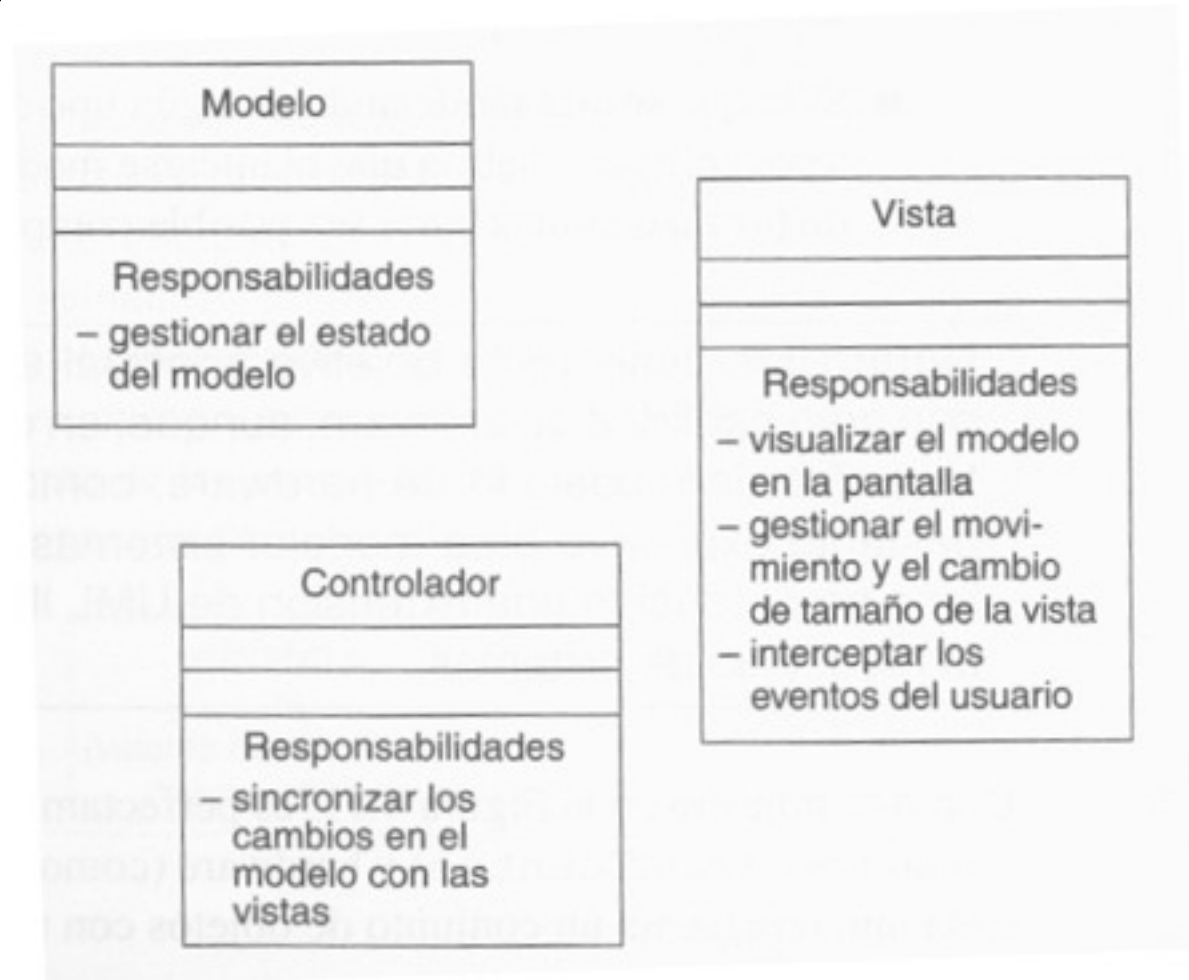
Modelado - Distribución de Responsabilidades

- **Distribución de Responsabilidades**
- Hay que conseguir un **equilibrio en el reparto de responsabilidades** porque:
 - Clases muy grandes son difíciles de cambiar y no muy reutilizables.
 - Clases muy pequeñas son difíciles de manejar y comprender.
- Para **modelar la distribución de responsabilidades**:
 1. Identificar un **conjunto de clases** que colaboran para que se realice cierto comportamiento.
 2. Identificar las **responsabilidades** de cada una.
 3. **Equilibrar** las clases de manera óptima:
 - Dividir las clases con demasiadas responsabilidades.
 - Agrupar clases con responsabilidades triviales.
 - **Redistribuir** las responsabilidades de manera adecuada.
 - Para que ninguna clase haga demasiado o muy poco en una colaboración.



Modelado - Distribución de Responsabilidades

- **Ejemplo.** Modelado con responsabilidades distribuidas de forma equilibrada.





Modelado – Semántica de una Clase

- **Semántica de una clase**
- Una vez identificadas las abstracciones (del problema o de la implementación de la solución), es necesario **especificar la semántica de las clases** que las representan.
- Para ello se distingue entre:
 - **Vista pública externa**
 - Especificación de lo que hace la clase.
 - Dirigida a los clientes.
 - **Vista privada interna**
 - Especificación de cómo lo hace.
 - Dirigida a los implementadores.
- UML dispone de un amplio rango de formas de modelar la semántica de una clase.



Modelado – Semántica de una Clase

- De menor a mayor nivel de formalidad, la semántica de una clase se puede especificar mediante:
 - - Las **responsabilidades**.
 - Un **texto estructurado en una nota estereotipada** (semantics) con junto a la clase.
 - Notas incluyendo el **cuerpo de cada método** como texto estructurado o en un lenguaje de programación. Cada nota se une a su operación por una dependencia.
 - Los **invariantes** de la clase y las **pre y post-condiciones** de cada operación. Se indican como texto estructurado en notas estereotipadas (invariant, precondition, postcondition) asociadas a la clase y operaciones mediante dependencias.
 - Una **máquina de estados** para la clase.
 - +
 - La **estructura interna** de la clase (nuevos diagramas en UML 2).
 - Una **colaboración** que representa a la clase.
 - **OCL** para expresar los invariantes de la clase y las pre y post-condiciones de cada operación.

F
o
r
m
a
l
i
d
a
d



Modelado - Colaboraciones

- **Colaboraciones**
- Ninguna clase se encuentra aislada.
 - Cada clase colabora con otras para llevar a cabo alguna semántica mayor que la suma de las semánticas individuales de cada clase.
 - **!El todo es mayor que la suma de las partes!**
- Además de capturar el vocabulario del sistema, es necesario modelar la forma en que los elementos del vocabulario colaboran entre sí.
 - Estas **colaboraciones** se representan mediante diagramas de clases.



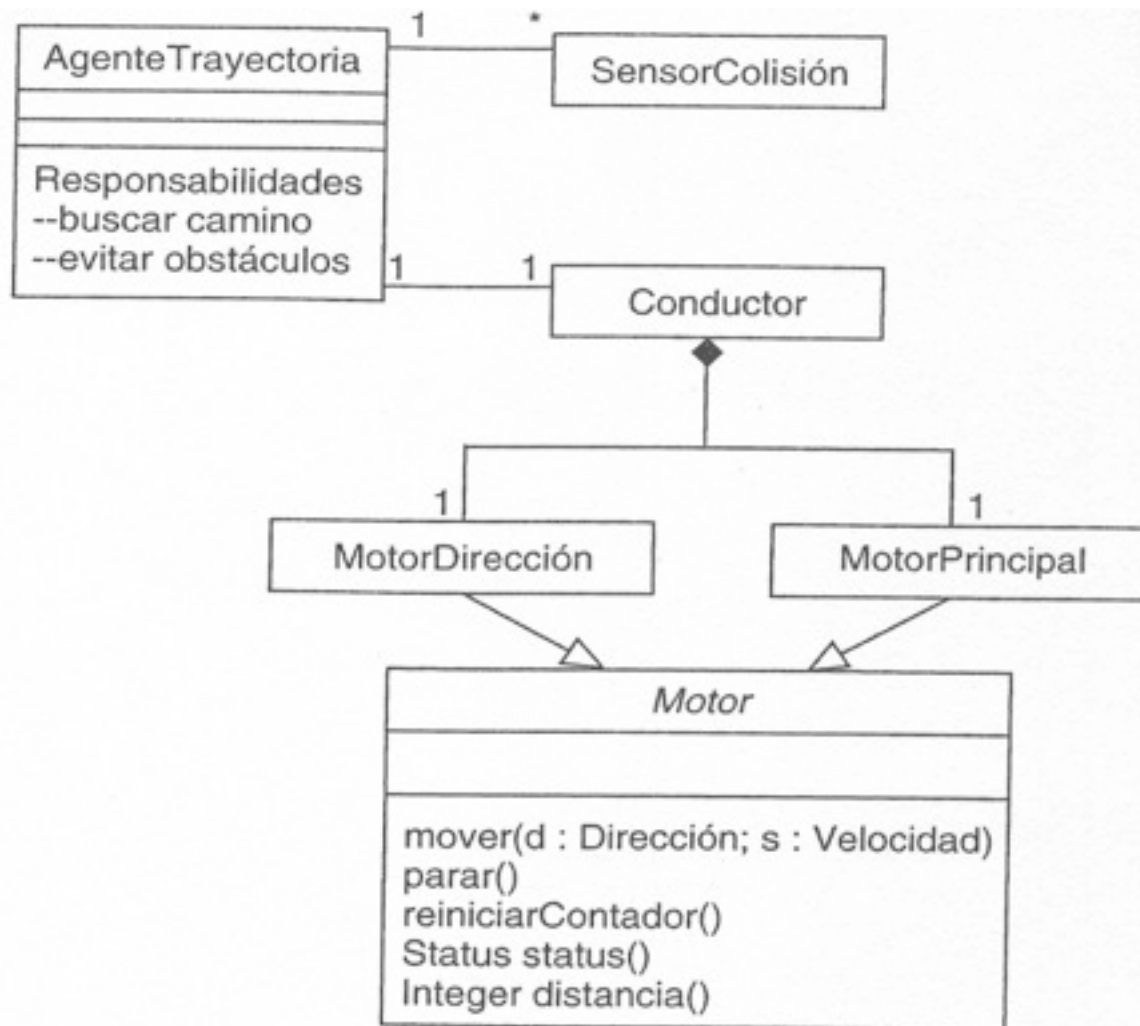
Modelado - Colaboraciones

- Para **modelar una colaboración**:
 1. **Identificar los comportamientos** de la parte del sistema que se quieren modelar.
 - Cada comportamiento es el resultado de la interacción de una sociedad de clases, interfaces y otros elementos.
 2. Para cada comportamiento, **identificar las clases, interfaces y otras colaboraciones** que intervienen.
 - Identificar también las **relaciones** entre dichos elementos.
 3. Para cada comportamiento, **usar escenarios** para recorrer la interacción entre los elementos.
 - Durante el recorrido se descubren partes que faltaban y otras que eran incorrectas.
 - Rellenar los elementos con su contenido: repartir las responsabilidades entre las clases, para después obtener los atributos y operaciones.



Modelado - Colaboraciones

- **Ejemplo.** Modelado de una colaboración para un robot autónomo.





Modelado – Esquemas de Datos

- **Esquemas de Datos**
- En muchos sistemas es necesario modelar **objetos persistentes** (objetos almacenados en un repositorio permanente o base de datos).
- **UML** permite modelar los tres tipos de **esquemas** que se manejan en **bases de datos**:
 - Conceptuales (vs ER, similares al modelado del negocio)
 - Lógicos (relacionales)
 - Físicos (para una tecnología concreta)
- Los diagramas de clases de UML son un superconjunto de los diagramas entidad-relación (ER).
 - UML \cong ER (datos) + Comportamiento



Modelado – Esquemas de Datos

- Para **modelar un esquema de datos**:
 1. Identificar las clases cuyo estado debe trascender el tiempo de vida de la aplicación (clases persistentes).
 2. Crear un diagrama de clases que contenga dichas clases.
 - Opcionalmente, definir un conjunto propio de valores etiquetados para detalles específicos de base de datos.
 3. Expandir los detalles estructurales de las clases persistentes:
 - Especificar atributos y asociaciones entre las clases, con multiplicidades.
 4. Crear abstracciones intermedias para simplificar la estructura y evitar patrones conflictivos.
 - Evitar o resolver ciclos de asociaciones.
 - Sustituir asociaciones uno a uno.



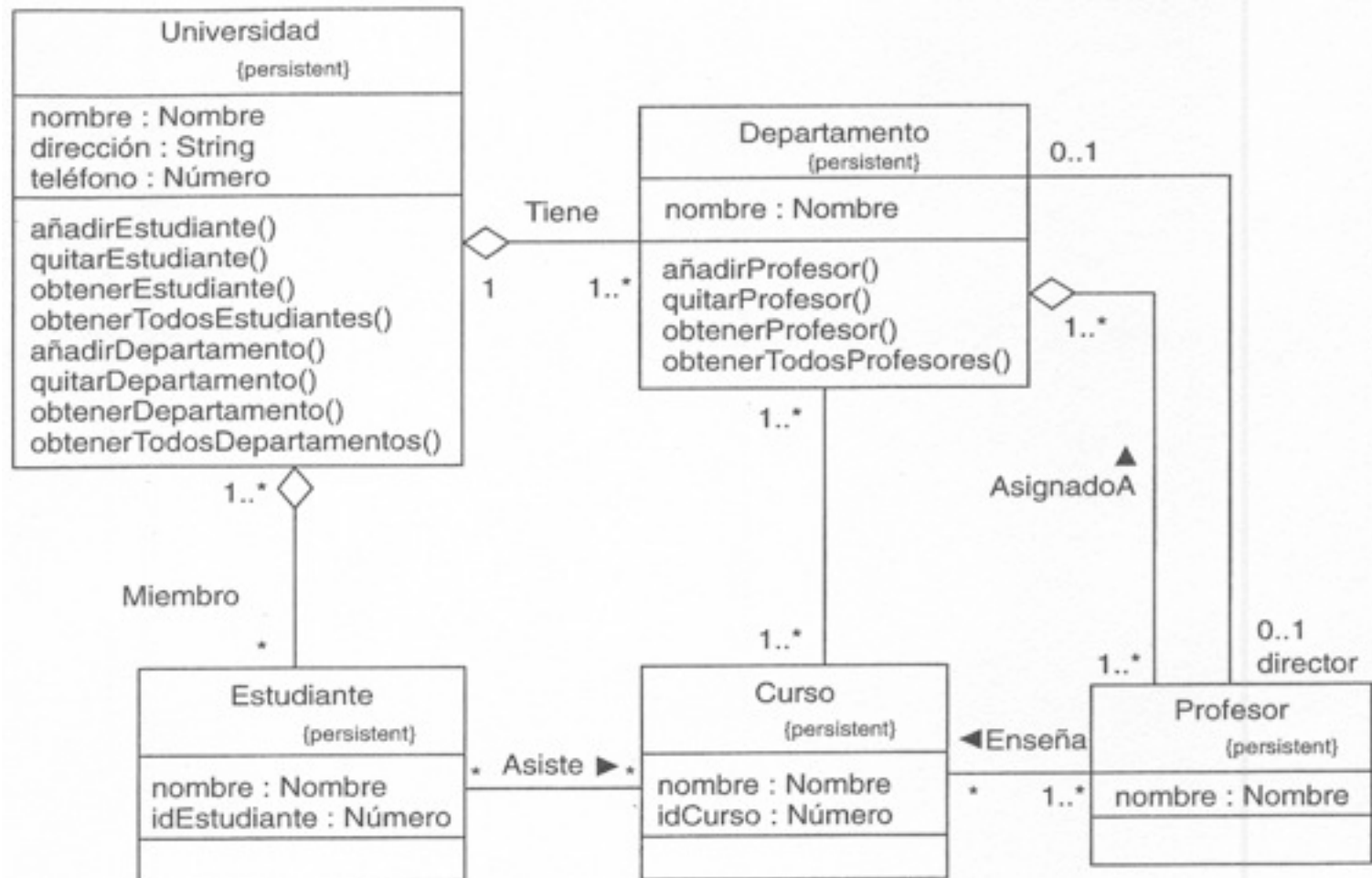
Modelado – Esquemas de Datos

- Para **modelar un esquema de datos**: (cont.)
 5. Considerar el comportamiento de las clases persistentes.
 - Expandir las **operaciones importantes** para el acceso a los datos y para la integridad de éstos.
 - Las reglas de negocio relativas a la manipulación de conjuntos de estos objetos persistentes deben encapsularse en una capa (paquete) por encima de las clases persistentes.
 - ESTILO EN TRES CAPAS (Interfaz, Dominio, Almacenamiento)
 6. Si es posible, usar herramientas que generen el esquema físico de forma (semi)-automática a partir del esquema lógico.
 - Visual Paradigm puede generar los esquemas SQL para diversos SGBDs.



Modelado – Esquemas de Datos

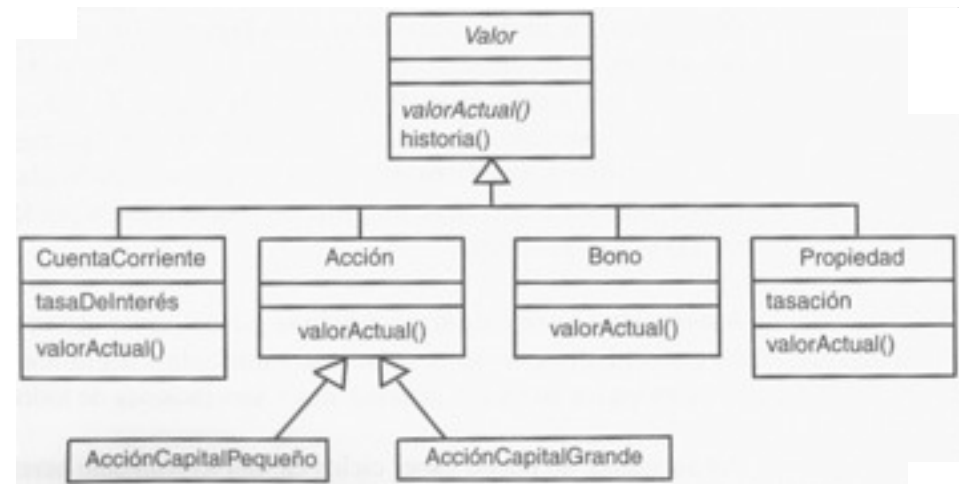
- **Ejemplo.** Modelo del esquema lógico de datos de un sistema de gestión universitaria.





Modelado – Redes de Relaciones

- **Redes de Relaciones**
- Para **modelar jerarquías de herencia**:
 - Buscar responsabilidades, atributos y operaciones comunes a dos o más clases.
 - Estos elementos comunes se adjudican a una clase más genérica.
 - Si no existe, dicha clase debe ser creada.
 - Hay que indicar que las clases más específicas heredan de la clase general





Modelado – Redes de Relaciones

- Las relaciones de **dependencia y generalización son asimétricas** al vincular dos clases de distinto nivel de importancia o abstracción.
 - En generalización, las clases hijas heredan de la clase padre, pero ésta última no tiene conocimiento de sus clases hijas.
- En cambio, al usar una **asociación es simétrica** porque conecta clases del mismo nivel:
 - Entre ambas clases existe un camino estructural (la asociación) a través del cual interactúan los objetos de las clases.



Modelado – Redes de Relaciones

- Para **modelar relaciones estructurales**:
 1. Para cada par de clases, hay que **especificar una asociación** entre ambas si:
 - a. Es necesario **navegar desde los objetos de una hacia los de la otra** (asociación guiada por los datos).
 - b. Los objetos de una necesitan **interactuar de alguna forma con los de la otra**, aparte de como variables locales de un procedimiento o parámetros de una operación (asociación guiada por el comportamiento).
 2. Para cada una de dichas asociaciones, especificar la **multiplicidad** y los **roles**.
 3. Si una de las clases es un todo comparada con las clases del otro extremo, que parecen sus partes, marcarla como una **agregación o composición**, según la exigencia de la semántica todo-parte.



Modelado – Redes de Relaciones

- **Ejemplo.** Modelado de relaciones estructurales.
 - ¿Qué asociaciones y generalizaciones ponemos?

Universidad

Departamento

Estudiante

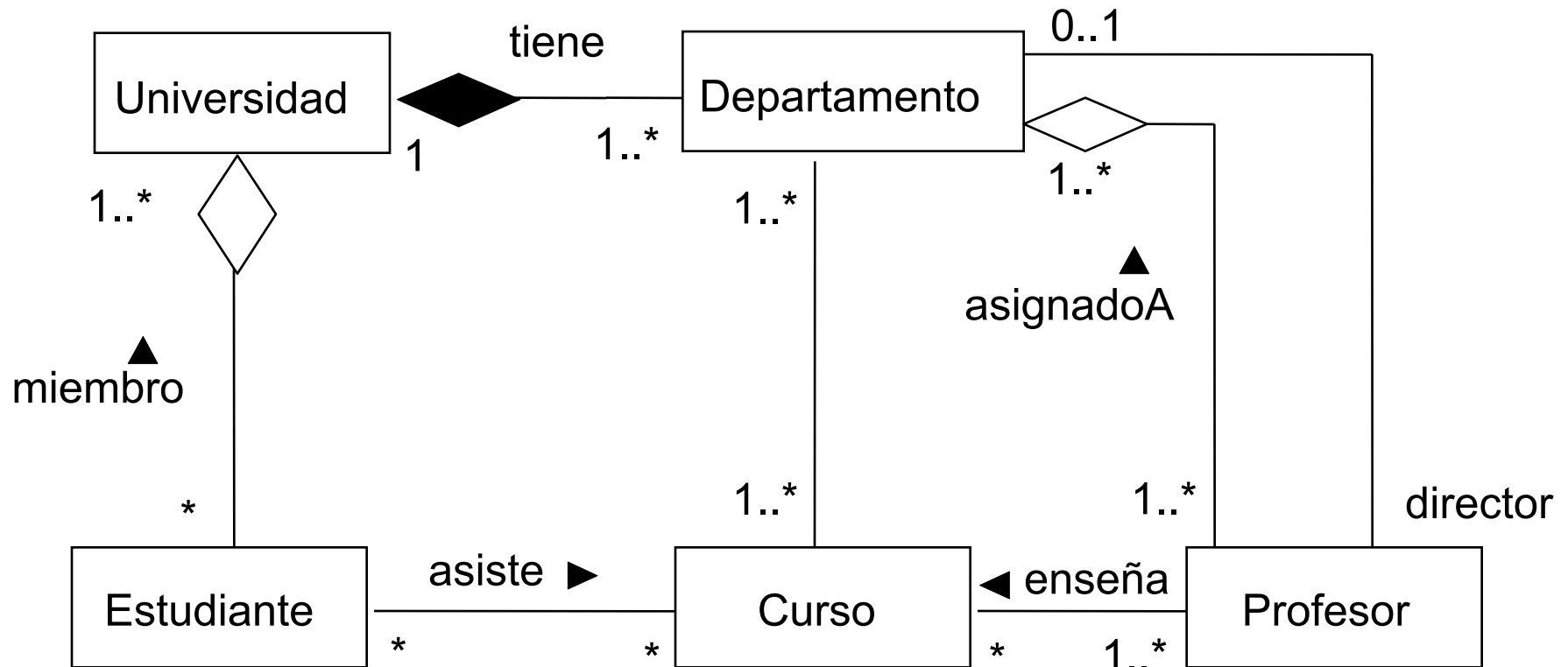
Curso

Profesor



Modelado – Redes de Relaciones

- **Ejemplo.** Modelado de relaciones estructurales.





Modelado – Redes de Relaciones

- Al **modelar redes complejas de relaciones** la clave del éxito es hacerlo de **manera incremental**, siguiendo los siguientes pasos:
 - Usar los **casos de uso y escenarios** para guiar el descubrimiento de las relaciones.
 - Modelar las relaciones estructurales mediante **asociaciones** (parte estática).
 - Identificar las relaciones de **generalización/especificación**.
 - Buscar **dependencias**.
 - Para cada relación, comenzar con las **características** básicas para después aplicar las avanzadas si es necesario.
 - Evitar diagramas muy complejos. Hacer **diferentes vistas** (diagramas) resaltando en cada uno conjuntos interesantes de relaciones.



Modelado – Líneas de Separación

- **Líneas de Separación**
- Las **interfaces** se utilizan para modelar las líneas de separación de un sistema compuesto de componentes software (Eclipse, .NET, JavaBeans, ..).
- Identificar las **líneas de separación** en un sistema implica identificar líneas claras de demarcación **en la arquitectura**. A ambos lados de estas líneas se encontrarán componentes que pueden cambiar de forma independiente.
- Al **reutilizar componentes** se suele disponer de un conjunto de operaciones y, quizás, alguna documentación pobre sobre ellas. Es necesario construir un modelo conceptual de su interfaz.
- Al **crear componentes** propios hay que comprender su contexto, especificando las interfaces en las que se basa y las interfaces que presenta al exterior.



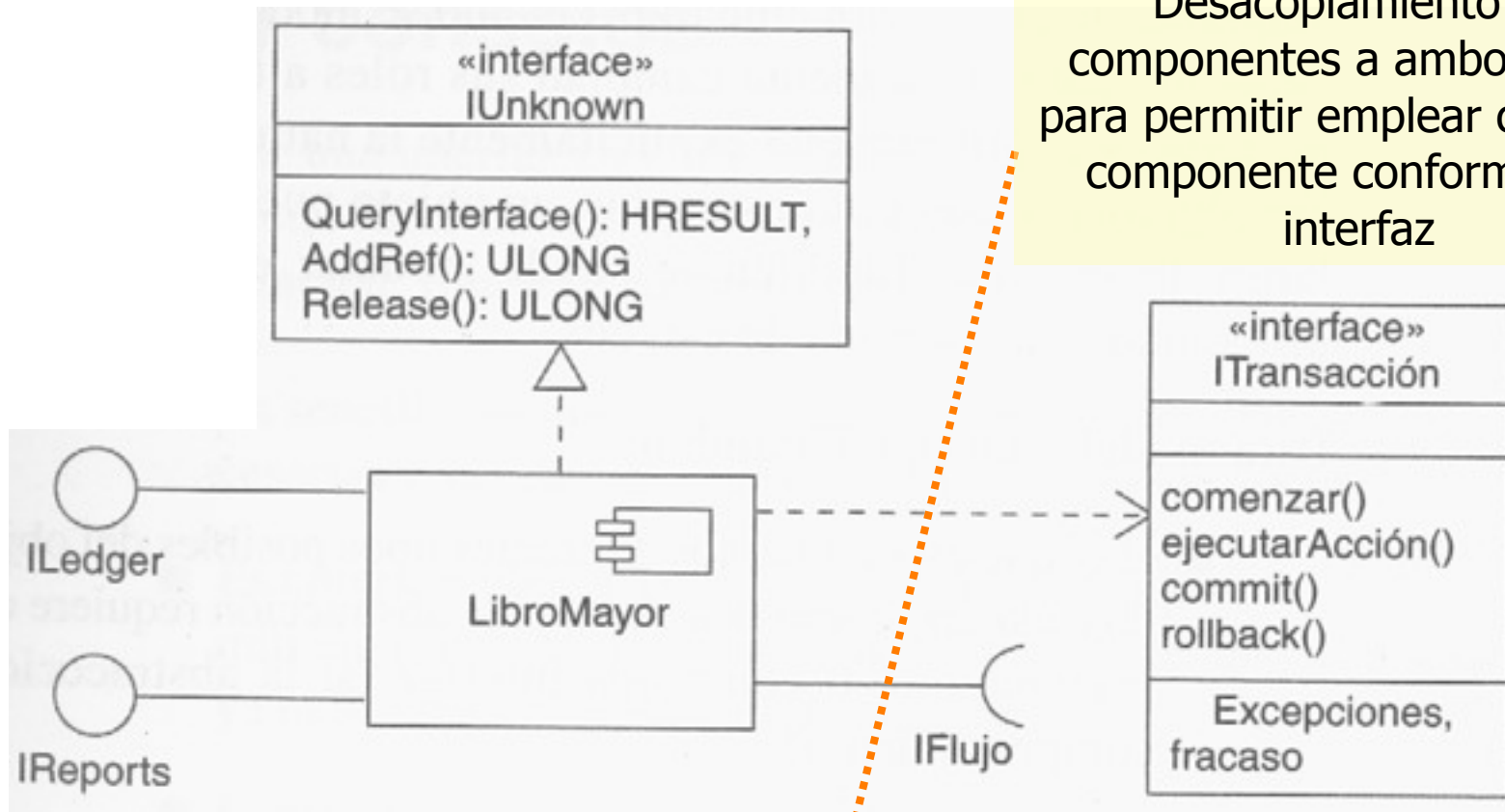
Modelado – Líneas de Separación

- Para **modelar las líneas de separación de un sistema**:
 1. En la colección de clases y componentes del sistema, dibujar una **línea** alrededor de los que tienden a acoplarse estrechamente con otros conjuntos de clases y componentes.
 2. Refinar las agrupaciones poniendo como **colaboraciones** las clases o componentes que tienden a cambiar juntos.
 3. Establecer las **operaciones** (y las señales) que cruzan estos límites.
 4. Empaquetar como **interfaces** los conjuntos relacionados lógicamente de estas operaciones y señales.
 5. Para cada colaboración, identificar las interfaces que requiere (importa) y las que suministra a otros (exporta).
 - Modelar la **importación** como dependencias y la **exportación** como realizaciones.
 - Para cada interfaz, documentar su dinámica mediante **pre y post-condiciones** para cada operación, y **casos de uso y máquinas de estados** para la interfaz como un todo.



Modelado – Líneas de Separación

- **Ejemplo.** Modelado de las líneas de separación de un componente en un sistema financiero.



Desacoplamiento de componentes a ambos lados para permitir emplear cualquier componente conforme a la interfaz



Modelado – Instancias

- **Instancias**
- Para **modelar instancias concretas**:
 1. Identificar las instancias que son necesarias y suficientes para modelar el problema.
 2. Representarlas en UML, asignándoles un nombre si es posible.
 3. Añadir el estereotipo, valores etiquetados y atributos (con sus valores) de cada instancia que son necesarios para modelar el problema.
 4. Representar dichas instancias y sus enlaces en un diagrama apropiado al tipo de instancia:
 - Diagrama de Objetos para instancias de clase.
 - Diagrama de Componentes para instancias de componente.
 - Diagrama de Despliegue para instancias de nodos.



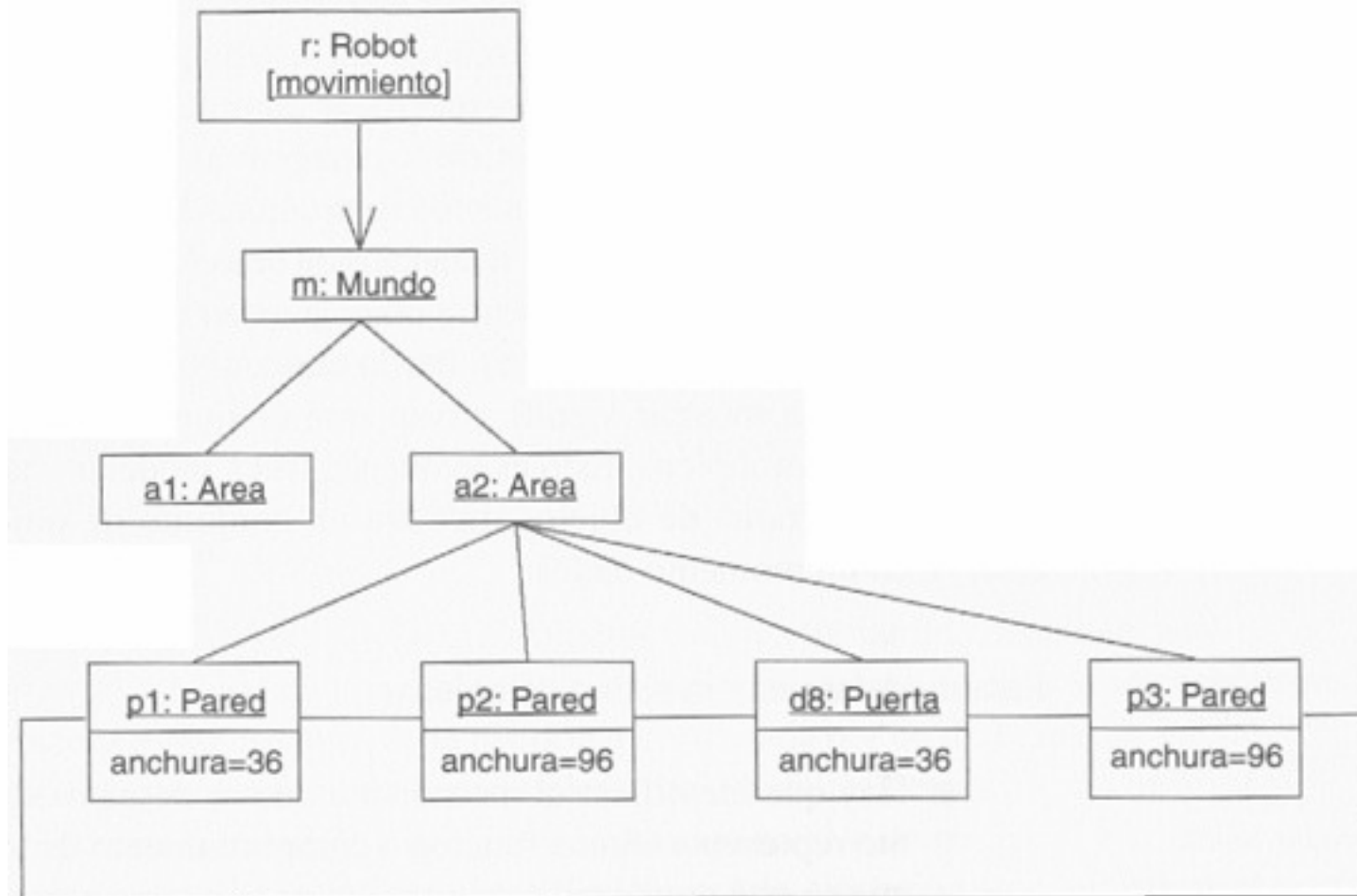
Modelado – Instancias

- Los **diagramas de objetos** sirven para mostrar estructuras de objetos, es decir, conjuntos interesantes de objetos concretos o prototípicos, relacionados entre sí.
- Para **modelar una estructura de objetos**:
 1. Identificar el comportamiento que se desea modelar.
 2. Crear una colaboración para describirlo.
 3. Identificar las clases, interfaces y demás elementos y las relaciones entre ellos.
 4. Considerar un escenario y congelarlo (foto fija), representando cada objeto que participa en el.
 5. Mostrar el estado y los valores de los atributos de los objetos, si es necesario para comprender el escenario.
 6. Mostrar los enlaces (instancias de asociación) entre los objetos.



Modelado – Instancias

- **Ejemplo.** Modelado de una estructura de objetos para un robot autónomo.





Mecanismos de Extensión

7. Mecanismos de Extensión

- Al presentar UML ya se indicaron los tres mecanismos disponibles para extender y particularizar UML 2:
 - **Estereotipos**
 - Extienden el vocabulario de UML, permitiendo definir nuevos tipos de elementos y relaciones a partir de los existentes pero específicos de un problema o dominio.
 - Algunos de uso frecuente ya están predefinidos en UML.
 - **Valores Etiquetados**
 - Extienden las propiedades de un estereotipo, permitiendo crear nueva información en la especificación del estereotipo.
 - **Restricciones**
 - Especifican condiciones sobre los elementos del modelo.



Mecanismos de Extensión

- Estos mecanismos, junto con las notas, permiten **especificar nueva semántica** no proporcionada con los elementos estándares predefinidos en UML.
 - =>
 - Permiten extender el “**Metamodelo**” de UML.
 - Mi UML
 - UML a la carta
 - UML con regla del 80/20
 - Hacen que la dicotomía UML vs DSLs se reduzca



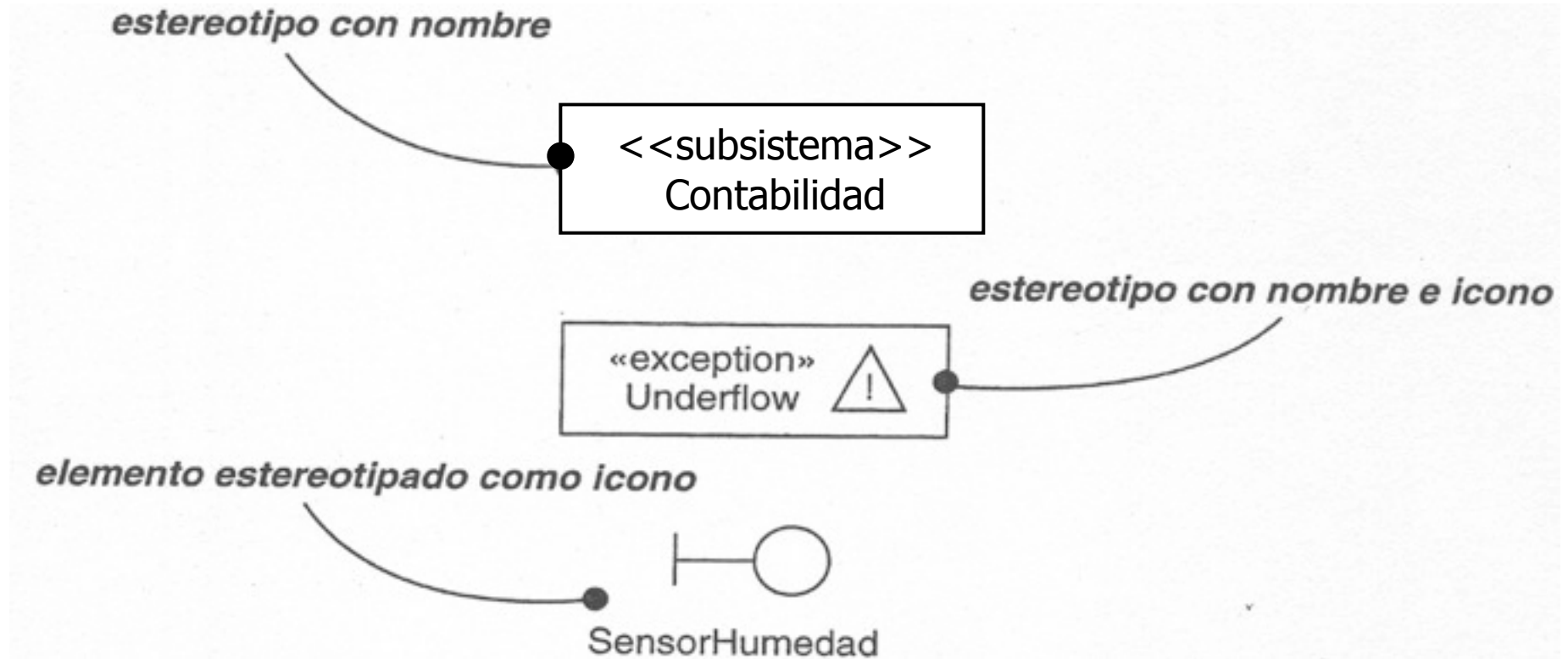
Mecanismos de Extensión - Estereotipos

- **Estereotipos**
- Es una **extensión del vocabulario de UML**:
 - Permite crear nuevos tipos de bloques de construcción similares a los existentes, pero específicos del problema que se modela.
- Se parte de un concepto UML (representado en su metamodelo), y se extiende, incorporándole aspectos diferenciadores:
 - Conjunto propio de propiedades (valores etiquetados)
 - Semántica propia (restricciones)
 - Notación diferenciada (icono)



Mecanismos de Extensión - Estereotipos

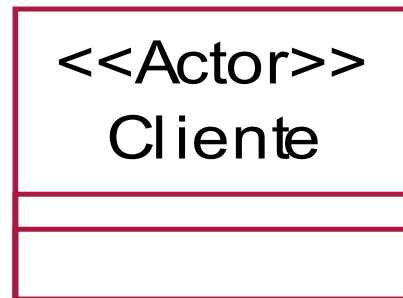
- **Ejemplos**



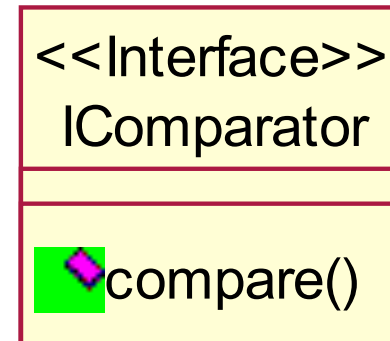


Mecanismos de Extensión - Estereotipos

- **Ejemplos** de Estereotipos Predefinidos



Cliente



IComparator

Clases estereotipadas



Mecanismos de Extensión - Estereotipos

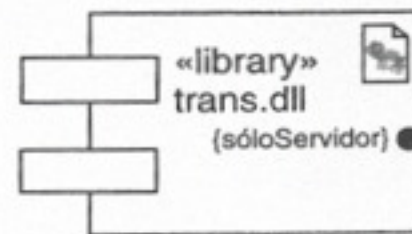
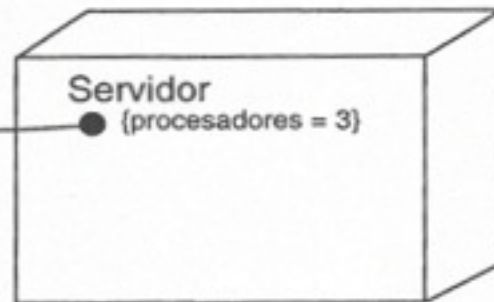
- Para **crear un estereotipo**:
 1. Asegurarse que el concepto no existe.
 2. Identificar el constructor de UML más cercano (clase, interfaz, componente, etc. – del metamodelo de UML).
 3. Definir el estereotipo a partir de dicho constructor, con todos sus detalles:
 - Las propiedades y semántica del nuevo elemento se definen mediante un conjunto de valores etiquetados y restricciones.
 - Si se desea asociar un símbolo gráfico hay que definir un icono.



Mecanismos de Extensión – Valores Etiquetados

- **Valores Etiquetados**
- Son una extensión de las propiedades de un elemento que permite añadir nueva información a su especificación.
 - Si con los estereotipos se pueden añadir nuevos constructores a UML, con los valores etiquetados se pueden añadir nuevas propiedades.

valor etiquetado

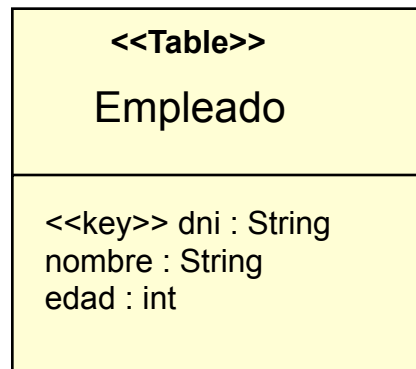


valor de la etiqueta

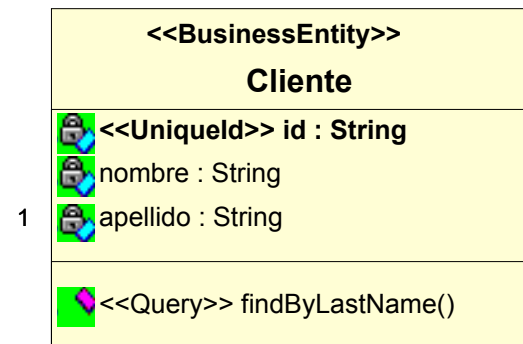


Mecanismos de Extensión – Valores Etiquetados

- No confundirlos con los atributos.



Estereotipo: Table
Valores Etiquetados: key

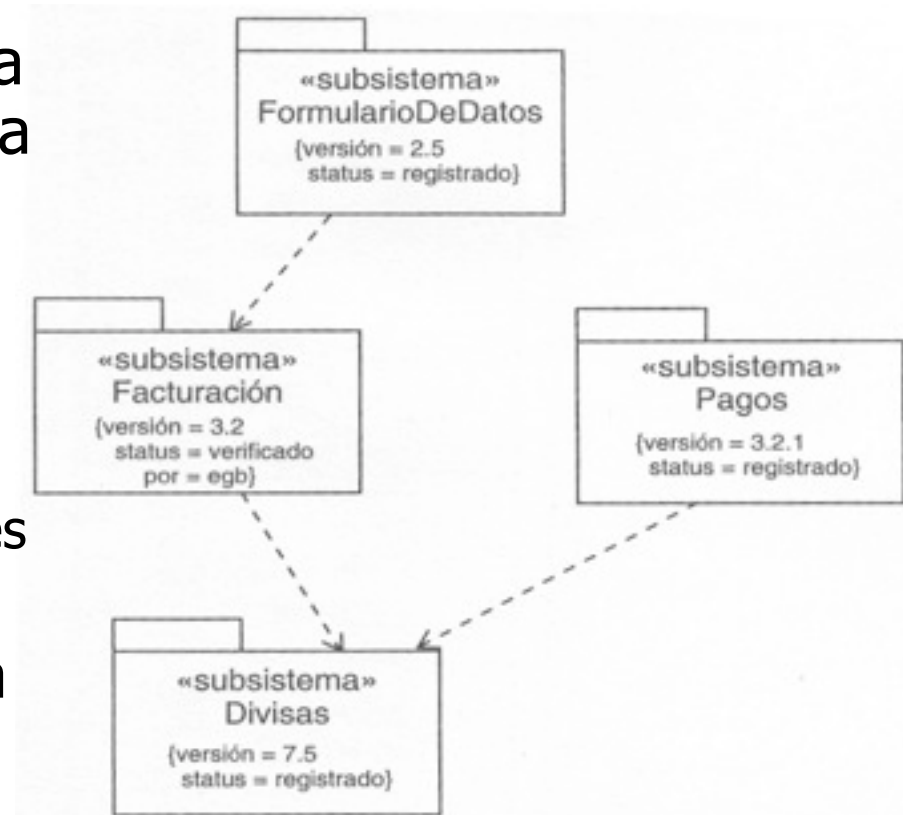


Estereotipo: BusinessEntity
Valores Etiquetados: UniqueID y Query



Mecanismos de Extensión – Valores Etiquetados

- Para **crear un valor etiquetado**:
 1. Asegurarse que no existe ya en UML nada que exprese la nueva propiedad que representa el valor etiquetado.
 - Tener en cuenta que la generalización (herencia) también se aplica a los valores etiquetados.
 2. Añadir el valor etiquetado a un elemento UML o a un estereotipo.





Mecanismos de Extensión – Restricciones


- **Restricciones o comentarios**
- Las **anotaciones** de UML permiten representar **restricciones o comentarios** asociados a un elemento o a una colección de elementos.
- No alteran el contenido semántico del elemento al que se asocian.

Publicar este componente en el repositorio del proyecto después de la próxima revisión de diseño. *egb 5/1/98*

texto simple

Véase <http://www.gamelan.com> para un ejemplo de este *applet*.

URL incluido

 Véase *encrypt.doc* para los detalles de este algoritmo.

enlace a un documento



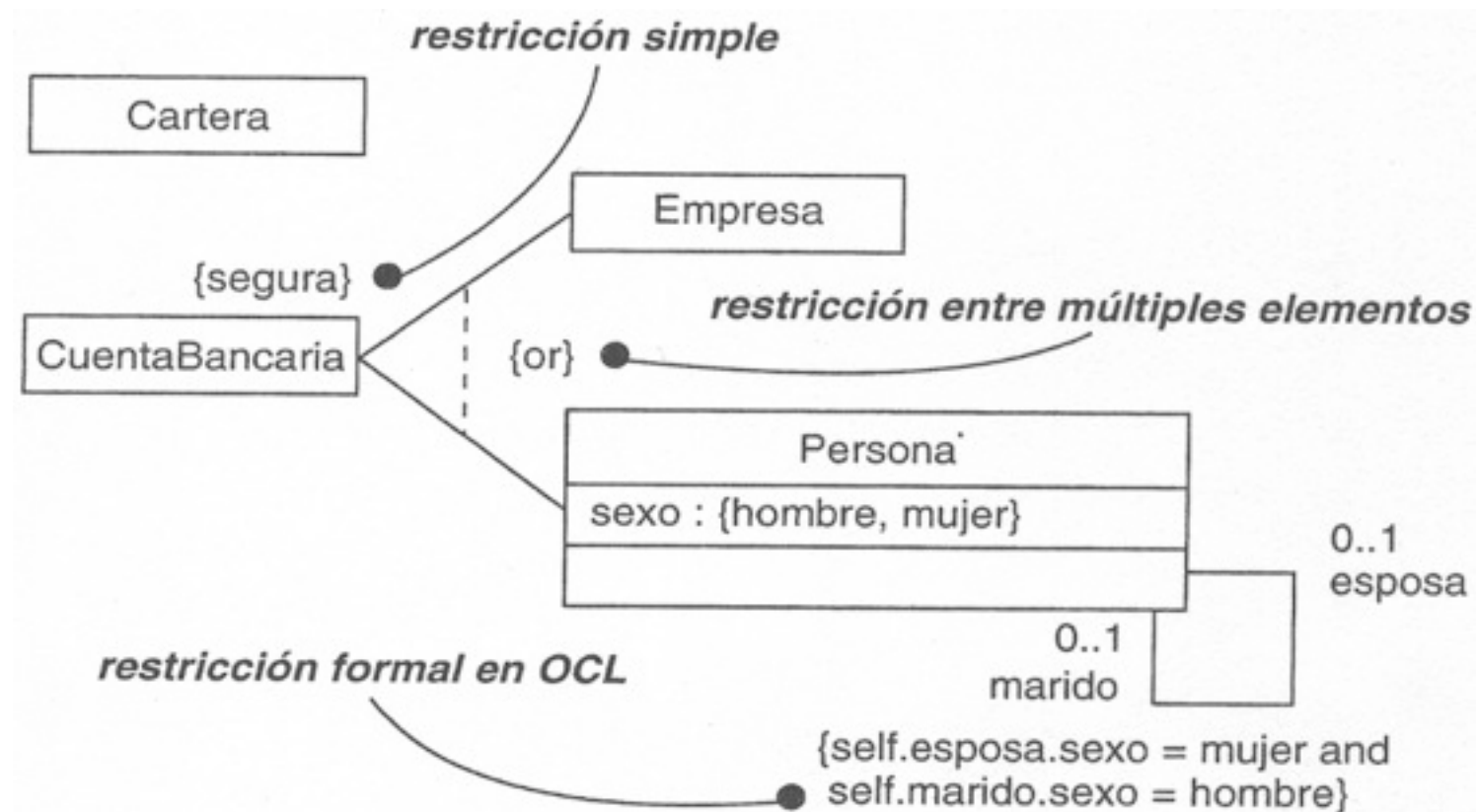
Mecanismos de Extensión – Restricciones

- Las **restricciones** extienden la semántica de un elemento añadiendo nuevas reglas o modificando las existentes.
 - Se representan con una **cadena de caracteres entre llaves**
 - a) Colocada junto al elemento al que está asociada o conectada, o
 - b) Conectada a ese elemento/s por relaciones de dependencia.



Mecanismos de Extensión – Restricciones

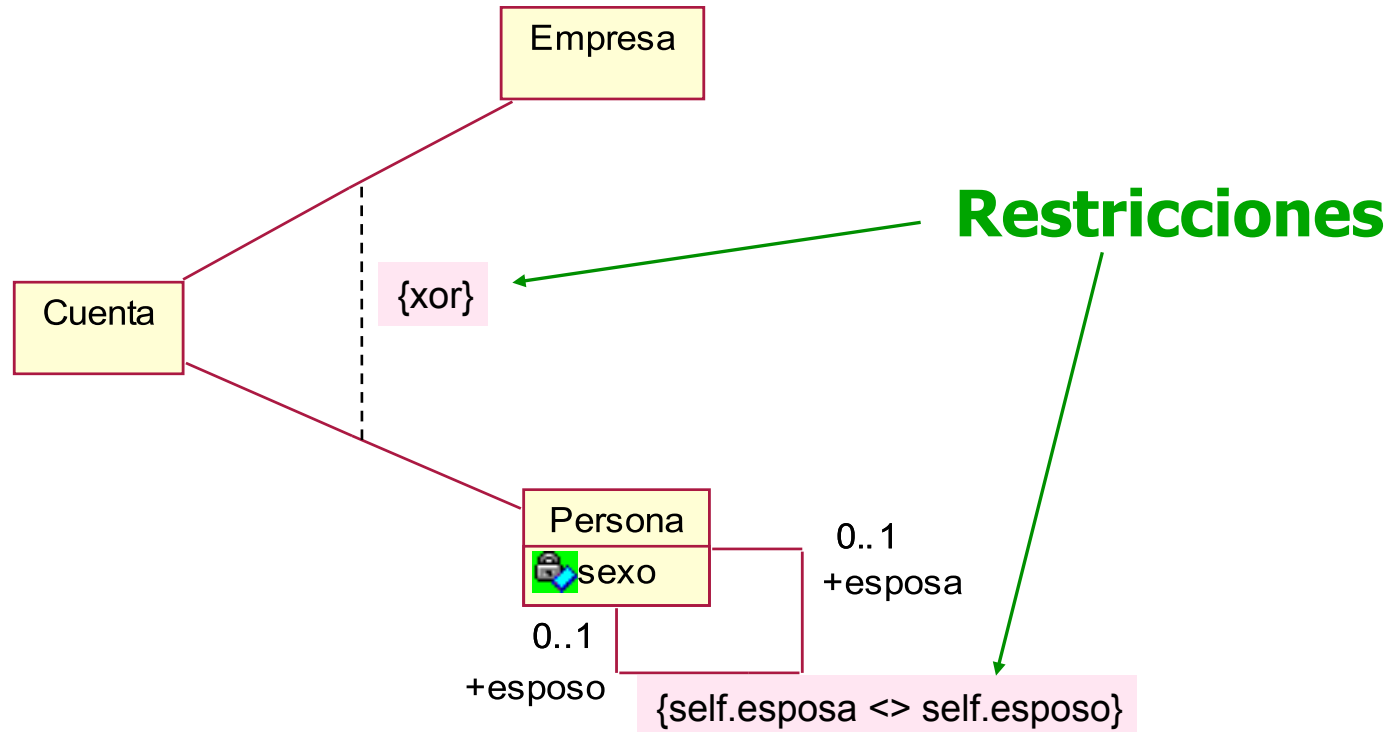
- Ejemplo de **restricciones**.





Mecanismos de Extensión – Restricciones

- Ejemplo de **restricciones**.





Mecanismos de Extensión – Restricciones

- Las restricciones se pueden formalizar en **OCL** (Object Constraint Language)
- OCL permite definir restricciones que no se pueden expresar en diagramas UML. Ejemplo:
 - “Dos tablas de un mismo esquema relacional deben tener distinto nombre”.

context Table

inv: tablasDistintoNombre

tablas -> forAll (t1, t2 |

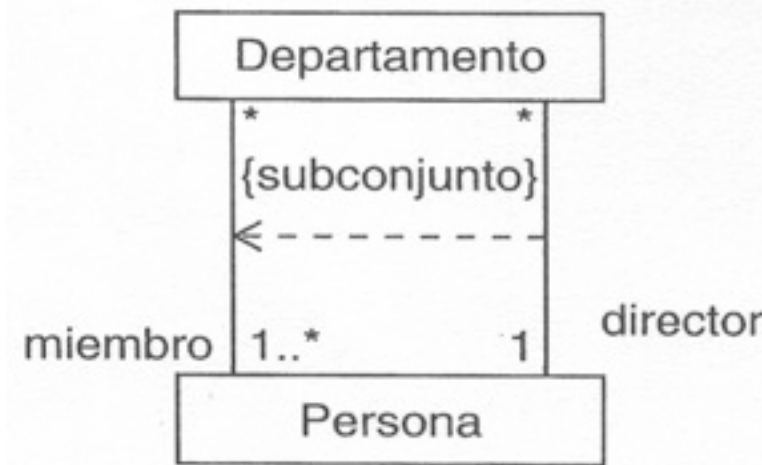
t1.name = t2.name implies t1 = t2)

end



Mecanismos de Extensión – Restricciones

- Para **crear una restricción**:
 - Asegurarse de que no existe ya en UML una forma de expresar la restricción.
 - Si se puede expresar en un diagrama
 - Escribirla como anotación adjunta (conectada) al elemento al que se refiere.
 - En caso contrario, escribirla en OCL aparte.



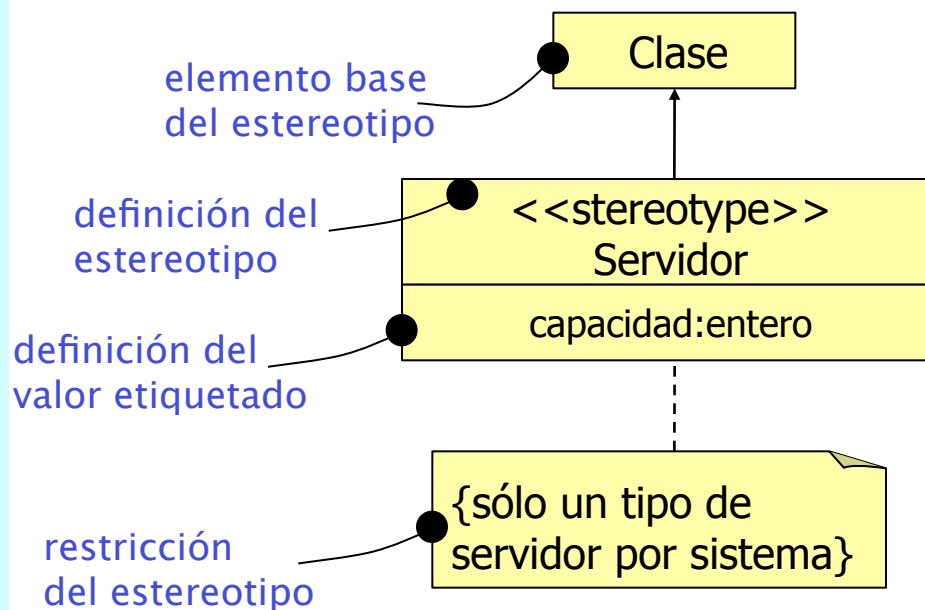


Mecanismos de Extensión

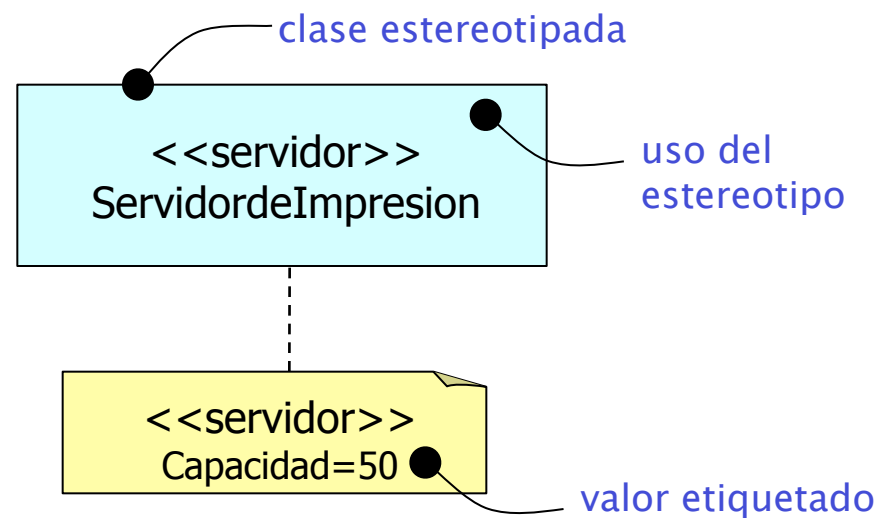
- **Ejemplo:**

- Un estereotipo "Servidor" a partir de "Clase", añadiéndole la propiedad (valor etiquetado) Capacidad (entero) y la restricción de que sólo puede haber un tipo de servidor por sistema.

Definición



Uso





Mecanismos de Extensión – Perfiles

- **Perfiles**
- Gracias a los mecanismos de extensión, UML se puede considerar como una familia de lenguajes de modelado:
 - Lenguaje UML core + perfiles
- Un **perfil** define una extensión y especialización de UML.
 - Implica una **forma específica de usar UML** para un dominio concreto (modelado del negocio, BBDD relacionales, ..).



Mecanismos de Extensión – Perfiles

- Un **perfil** está formado por un conjunto predefinido de:
 - Estereotipos
 - Con su correspondiente notación – iconos
 - Valores Etiquetados
 - Restricciones
 - Clases básicas
 - Subconjunto de tipos de elementos de UML
 - Evitar confusiones con tipos de elementos no necesarios
 - Ejemplo: En BBDD no incluir “nodo”.



Mecanismos de Extensión – Perfiles

- Algunos de los perfiles existentes:
 - Systems Modeling Language (SysML)
 - for Data Distribution
 - for Modeling and Analysis of Real-time and Embedded Systems (MARTE)
 - for Modeling QoS and Fault Tolerance Characteristics and Mechanisms
 - for Software Radio
 - for Voice
 - ...