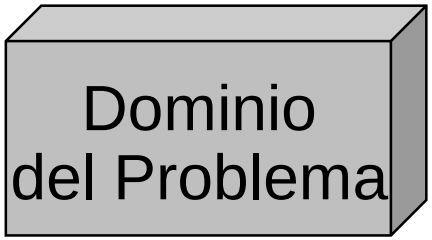
Fase de análisis – Modelado de objetos

Índice

- 1. Introducción
- 2. Análisis
- 3. Modelado de objetos
  - Objetivos
  - Conceptos: objeto, clases, relaciones
- 4. Técnica de construcción del modelo de objetos

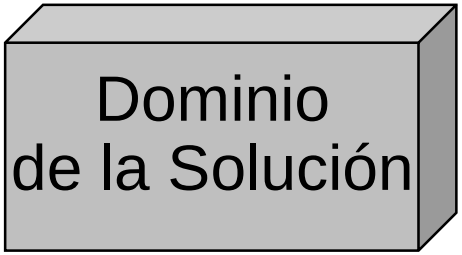
1. Introducción - Recordatorio



Análisis del problema
¿Cuál es el problema?

Análisis de la solución
¿Cuál es la solución?

Diseño de la solución
¿Cuál es el mecanismo que mejor implementa la solución?



Implementación de la solución
¿Cómo está el problema resuelto?

Prueba del producto
¿Está el problema resuelto?

Mantenimiento
¿Son necesarias alguna mejoras?

Transferencia
¿Puede el usuario utilizar la solución?

Recordatorio …

- Antes de construir algo (una casa, un puente, un programa) debemos entender los requisitos y el entorno en el cual va a instaurarse
- Fase de Requisitos:
  - Definición del sistema en términos entendidos por el cliente ("Descripción del problema")
- Fase de Análisis:
  - Especificación técnica del sistema en términos entendidos por el desarrollador ("Especificación del problema")

2. Análisis

- El análisis pretende obtener un *modelo* preciso, conciso, comprensible y correcto del mundo real (del problema que queremos resolver)
- El objetivo del análisis es por tanto construir modelos del sistema a desarrollar
- Es preciso abstraer primero las características importantes y dejar para más adelante los pequeños detalles
- Los modelos de análisis que tienen éxito indican lo que es preciso hacer, sin limitar cómo ha de hacerse
- El resultado del análisis es la comprensión y modelado del problema

Análisis …

- Partiendo del catálogo de requisitos, y con la ayuda del cliente, el análisis construye modelos que tienen que servir para refinar esos requisitos
- Además esos modelos son la base para el diseño, permitirán construir una solución
- Construir un modelo riguroso del dominio del problema obliga al ingeniero a enfrentarse a los errores de comprensión en una fase temprana, cuando todavía es fácil corregirlos

Análisis ...

- En el análisis orientado a objetos modelamos:
  - La estructura estática del sistema: Modelo de objetos
  - El comportamiento del sistema: Modelo dinámico
- El *modelo de objetos* describe la información que existe en el sistema y sus relaciones
- El *modelo dinámico* estudia las interacciones de los actores con el sistema y las interacciones de los objetos dentro del sistema

Análisis ...

- UML nos ofrece los siguientes diagramas para construir los modelos de análisis:
  - Modelo de objetos
    - Diagrama de objetos
    - Diagrama de clases
  - Modelo dinámico
    - Diagramas de interacción (colaboración)
      - Diagrama de secuencia
      - Diagrama de comunicación
    - Diagrama de estados
    - Diagrama de actividades

3. Modelado de objetos

- Objetivos
  - El modelo de objetos muestra la estructura estática de datos correspondiente al sistema del *mundo real*
  - Organiza la estructura mostrando *clases* de objetos del mundo real, sus atributos y relaciones entre las clases
  - Queremos organizar el dominio del problema en un conjunto de abstracciones ordenadas de forma que obtengamos un conocimiento más en profundidad del problema
  - Los diagramas del modelo de objetos promueven la comunicación analista-experto

Conceptos (coinciden con los vistos en Programación Orientada a Objetos)

- Objeto
- Clase: atributos y operaciones
- Relaciones entre clases
  - Asociación
  - Agregación/ Composición
  - Herencia

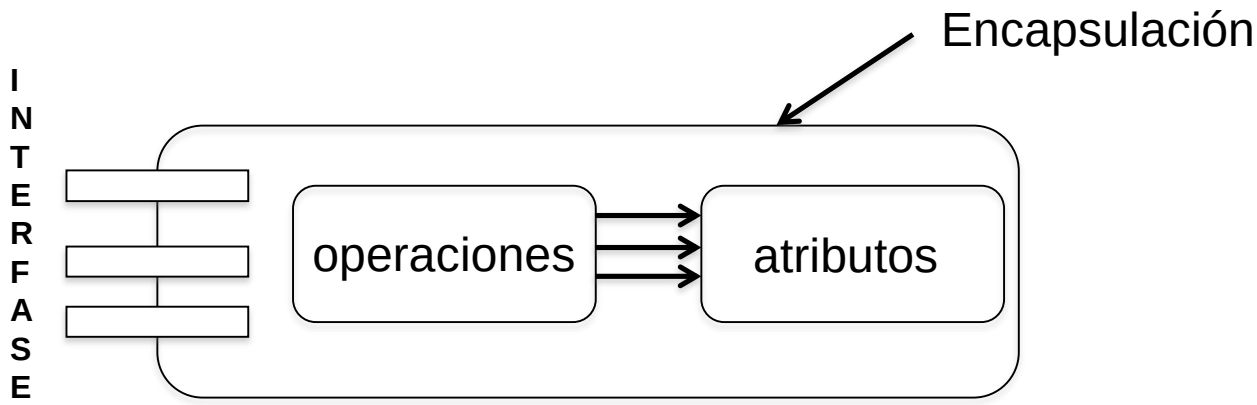
- Mecanismos de abstracción
  - Clasificación / Instanciación
  - Asociación / Individualización
  - Composición / Descomposición
  - Especialización / Generalización

Objetos

- El propósito del modelado de objetos es identificar los objetos que existen en el dominio del problema
- Antonio Pérez, Google, una mesa son objetos
- Un objeto es un concepto, abstracción o entidad física con límites bien definidos y con significado en el dominio del problema
- La descomposición de un problema en objetos depende de la naturaleza del problema y del juicio → No existe una única representación correcta

Objetos …

- Como se vio en Programación Orientada a Objetos un objeto tiene parte estática y parte dinámica (atributos y operaciones)
- Un objeto encapsula estado y comportamiento
- A los atributos del objeto no se puede acceder



Objetos …

- Ventajas de la encapsulación
  - Protección de datos
  - Disminución del acoplamiento entre clases
  - Favorece modularidad y mantenimiento

Objetos …

- El estado representa los valores actuales de las propiedades (atributos y enlaces) del objeto
- El estado del objeto evoluciona con el tiempo
- El estado del objeto recuerda el efecto de las operaciones sobre él mismo
- El *comportamiento* es el conjunto de operaciones del objeto
- El comportamiento nos explica las *competencias* del objeto: las acciones y reacciones que puede realizar

Objetos …

- Objeto = Estado + Comportamiento + Identidad
- Los objetos poseen identidad. Los objetos se distinguen por su existencia y no por las propiedades descriptivas que puedan tener
- En diseño e implementación hay que elegir una aproximación para realizar la identidad:
  - Identificadores
  - Variables de memoria
  - Claves en bases de datos

Objetos …

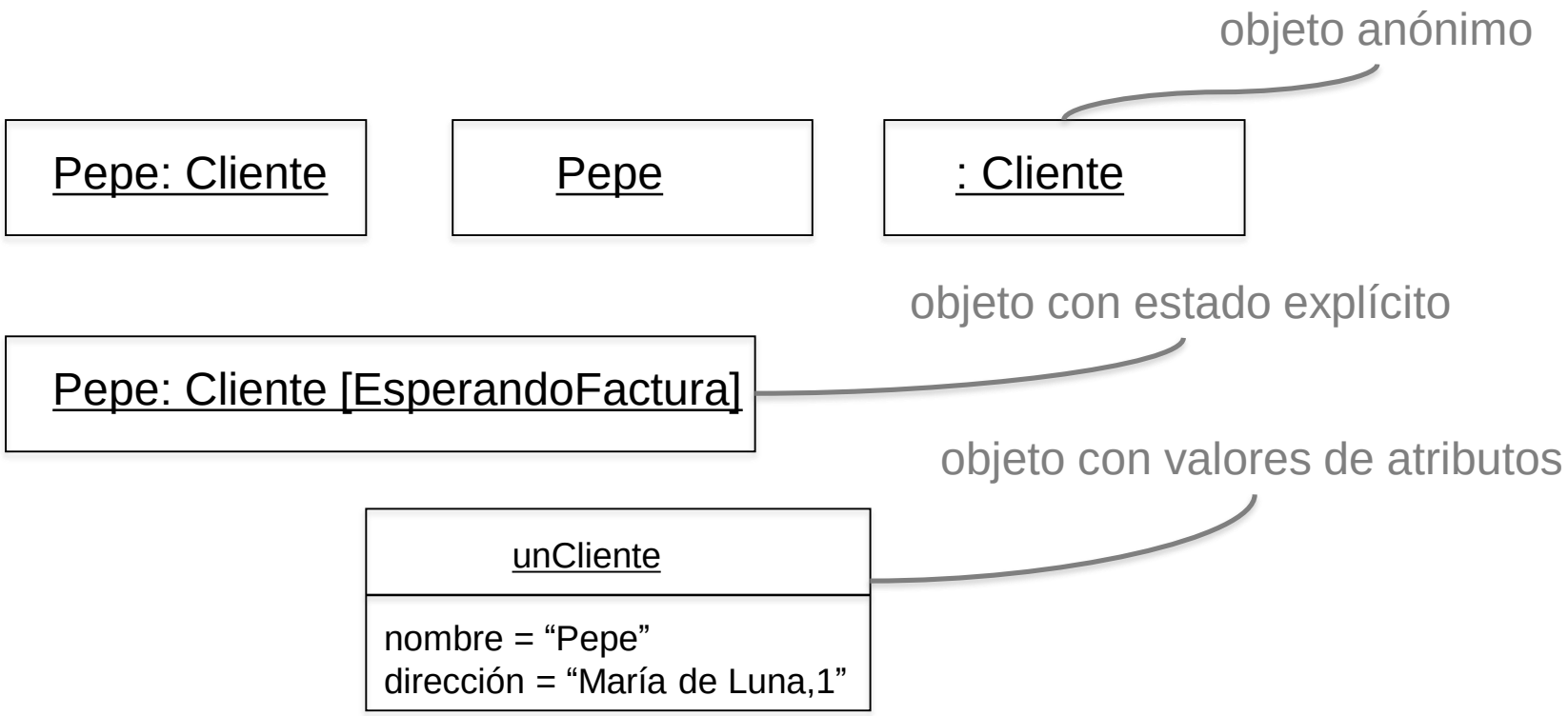
- Además, los objetos pueden ser *persistentes*
- Persistencia es la capacidad de trascender en el tiempo
- Los lenguajes de POO suelen dar un soporte muy básico a la persistencia (*serialización*)
- Se suele recurrir a artificios externos para garantizar persistencia

Diagramas de objetos

- Los diagramas de objetos representan objetos y sus *relaciones*
- Describen una instantánea estática de las instancias de los elementos existentes en el diagrama de clases
- Son útiles tanto para análisis como para **diseño**

Objetos …

- Representación de objetos en UML



Clases

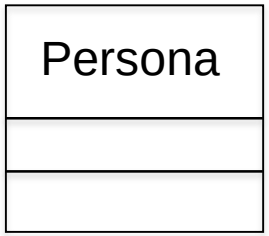
- Una clase describe un grupo de objetos que comparten propiedades (atributos y operaciones), restricciones, relaciones con otros y tienen una semántica común
- Persona, compañía o mesa son clases
- Los objetos y las clases suelen aparecer como sustantivos en el dominio del problema

Clases …

- Los objetos de una clase tienen los mismos atributos, relaciones y patrones de comportamiento
- Es posible tener, en el dominio del problema, objetos con valores de atributos idénticos y las mismas relaciones → *identidad*. En el dominio de la solución NO

Clases …

- *Caballo* y *casa* tienen coste y edad pero pueden pertenecer a clases diferentes.
  - Como *activos financieros* pertenecerán a la misma clase.
  - Pero si consideramos que las personas *alimentan* a los caballos y *pintan* las casas entonces pertenecen a clases diferentes → No existe una única representación correcta
- Representación de clases en UML:
  - nombre
  - atributos
  - operaciones



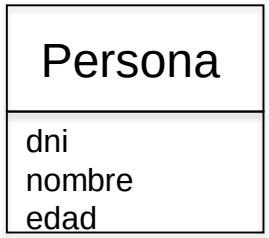
Atributos

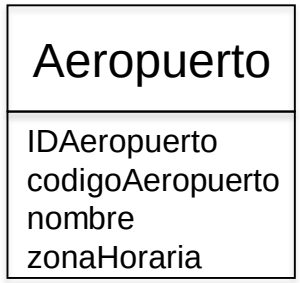
- Nombre, edad y peso son atributos de Persona. Describen propiedades de los objetos
- Cada atributo tiene un valor en cada instancia. Cuando tienen el mismo valor en todas las instancias se llaman *atributos de clase*
- El nombre del atributo es único dentro de la clase
- Los atributos tienen valores puros de datos (tipos básicos) y no objetos. (En **Diseño** será diferente)
- Los tipos básicos *no tienen identidad*

Atributos …

- En el modelo de objetos no deben aparecer atributos que actúen como identificadores, ya que los objetos tiene identidad *per se*
- Sin embargo no hay que confundir identificadores con atributos del mundo real. Por ejemplo: dni es un atributo del mundo real y por tanto puede aparecer en Análisis
- En Diseño sí aparecerán atributos identificadores para las clases

Buen modelo para Análisis y Diseño





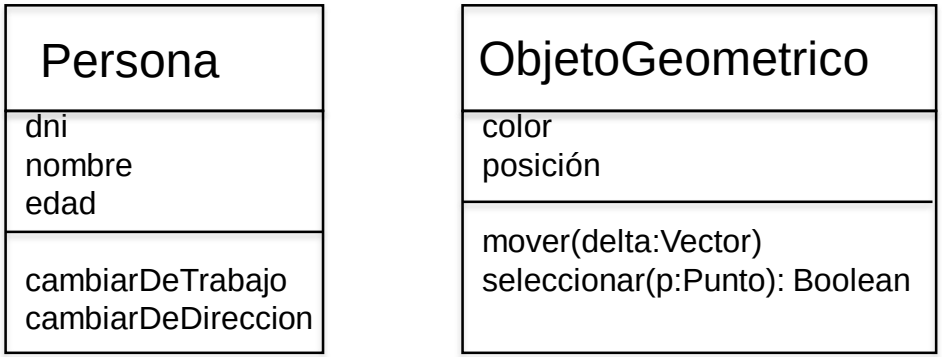
Mal modelo para Análisis, bueno para Diseño

Operaciones y métodos

- Una *operación* es una transformación que se puede aplicar por los objetos de una clase o ser aplicada a ellos
- Contratar, despedir, pagar dividendos son operaciones de Compañía. Cantar, reír y saltar de Persona
- Los objetos de una clase comparten las mismas operaciones
- Las operaciones poseen argumentos. El número de argumentos, su tipo y el tipo de retorno de la operación se llama *signatura*
- Una misma operación puede aplicarse en clases distintas → *operaciones polimórficas*

Operaciones y métodos …

- Un *método* es la implementación de una operación para una clase
- Hay operaciones que modifican el *estado* del objeto
- Otras no lo hacen, se llaman *consultas*
- Una consulta sin parámetros es un *atributo derivado*



Instanciación

- Es la operación que permite crear objetos
- En análisis cada clase declara una o varias formas (operaciones) para crear objetos y para destruirlos
- Los lenguajes de POO implementan dichas operaciones

Relaciones

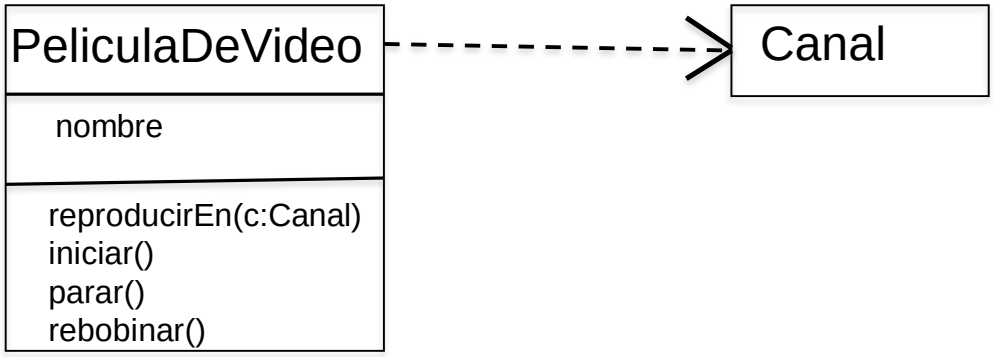
- Cuando modelamos un sistema nos damos cuenta de que muy pocas clases se encuentran aisladas. La mayoría colaboran (se relacionan) con otras.
- Tipos de relaciones
  - Dependencia:
    - Es una relación de uso
  - **Asociación**
    - Es una relación estructural
    - Refinamiento de las asociaciones: **agregación, composición**
  - **Herencia**
    - Es una relación de generalización/especialización
  - Realización (la veremos en el tema de diseño de objetos)
    - En el contexto de las interfaces y de las colaboraciones

Dependencia

- Es más un concepto de **Diseño** que de **Análisis**
- Una dependencia indica que un *elemento* usa a otro (una clase usa otra clase)
- También se crean dependencias entre otros elementos (paquetes, componentes, artefactos, …)
- Suele indicar que una clase utiliza las operaciones de otra o tiene parámetros cuyo tipo es la otra clase
- Las dependencias suelen tener estereotipos

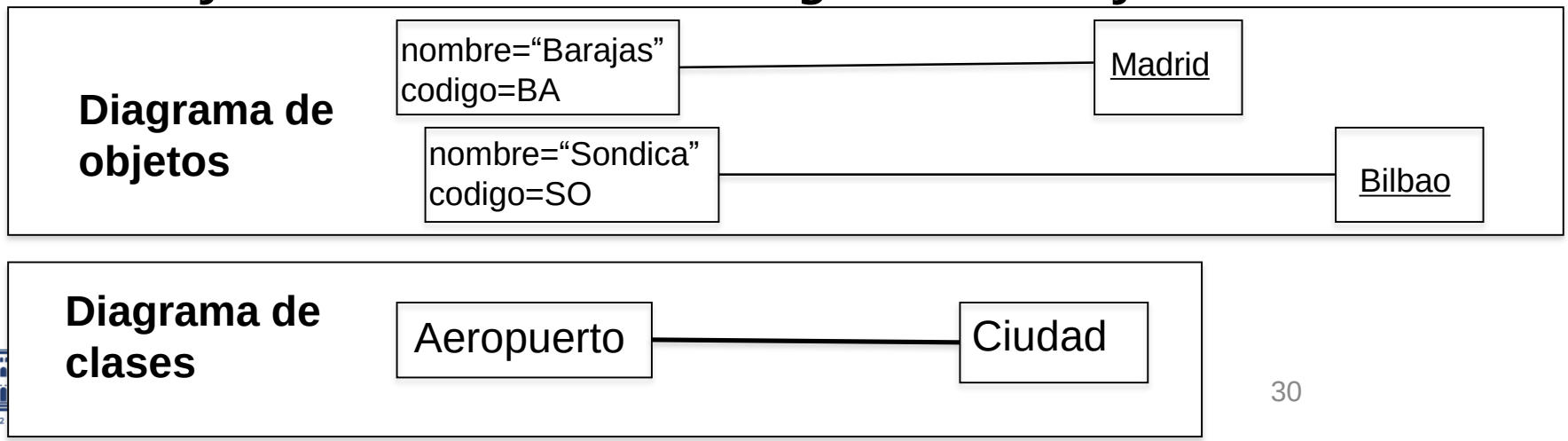
Dependencia …

- Un cambio en la especificación de un elemento afectará al que lo usa, no necesariamente a la inversa



Asociación

- Una asociación es una relación que indica que existe una conexión física o conceptual entre objetos
- La asociación es una relación *estructural*
- En el **diagrama de objetos** la relación se muestra como "enlaces" entre los objetos
- En el **diagrama de clases** una asociación abstrae un conjunto de enlaces del diagrama de objetos

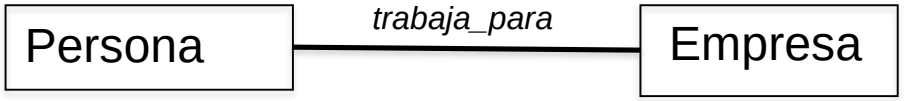


Asociación …

- Conceptos básicos
  - Nombre de la asociación
  - Roles
  - Multiplicidad

Asociación ...

- Nombre de la asociación
  - La asociación puede tener un nombre que describa la naturaleza de la relación



Roles

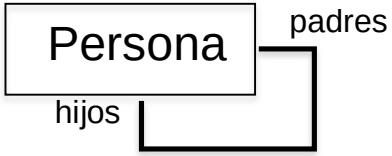
- Se puede expresar explícitamente el rol que juega una clase en la asociación
- El rol es la cara que una clase presenta a la clase del otro extremo de la asociación

- Una clase puede jugar el mismo o diferentes roles en otras asociaciones
- Los roles se pueden ver como una alternativa al nombre de la asociación
- Los roles pueden coexistir con el nombre de la asociación

Roles ...

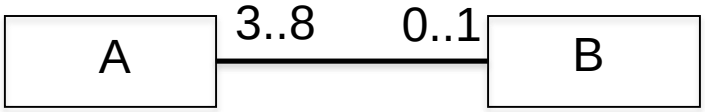
- Los roles se utilizan para recorrer las asociaciones. Se ven como pseudoatributos
  - unaPersona.empleador
- Los roles son opcionales si el modelo no es ambiguo
- Si hay varias asociaciones entre las clases entonces hay que usar roles y/o nombres de asociación para distinguirlas

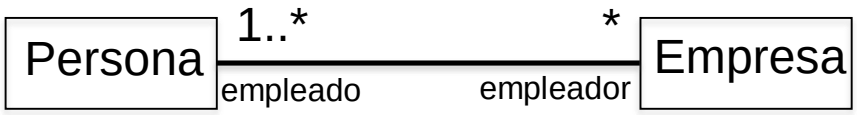
- Los roles son obligatorios en las asociaciones reflexivas



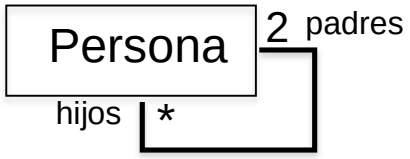
Multiplicidad

- Indica cuántos objetos de una clase se relacionan con una única instancia de la otra clase de la asociación
- Lectura: Dado un objeto de la clase A ¿con cuántos objetos, como mínimo y como máximo, de la clase B se relaciona?
- Representación: *valorMin .. valorMax*
- Ejemplos:





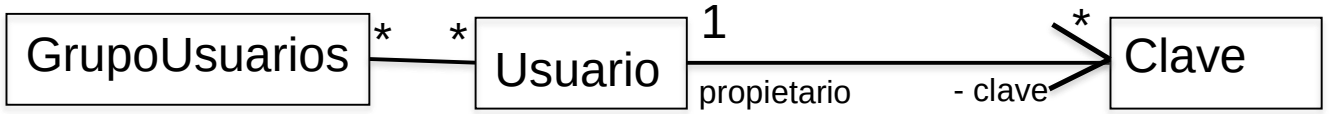
- Si no se indica nada la multiplicidad es 1..1
- 1 es equivalente a 1..1
- * es equivalente a 0..*



Navegación y Visibilidad

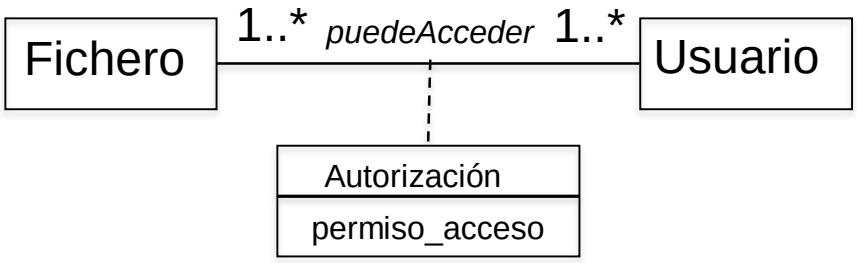
- En la fase de **Análisis** todas las asociaciones son consideradas bidireccionales
- La navegación indica en qué sentido/s se recorre la asociación, es por tanto un concepto de **Diseño**
- La navegación indica que un objeto puede llegar a los objetos del otro extremo

- La *visibilidad* también es un concepto de **Diseño** y permite que los objetos no sean accesibles a ningún objeto externo a la asociación



Clases asociación

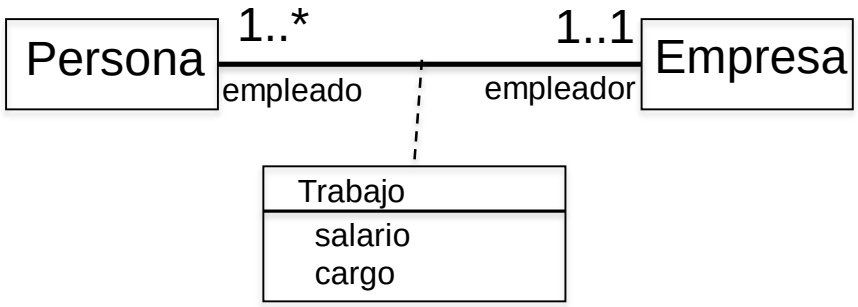
- En una asociación entre dos clases la propia asociación puede tener propiedades
- Esto se modela en UML con una clase asociación
- Una clase asociación tiene propiedades de clase y propiedades de asociación
- Cada enlace de la asociación tiene una instancia distinta de la clase asociación
- La notación es una línea discontinua



- No pueden situarse en una clase sin pérdida de información

Clases asociación ...

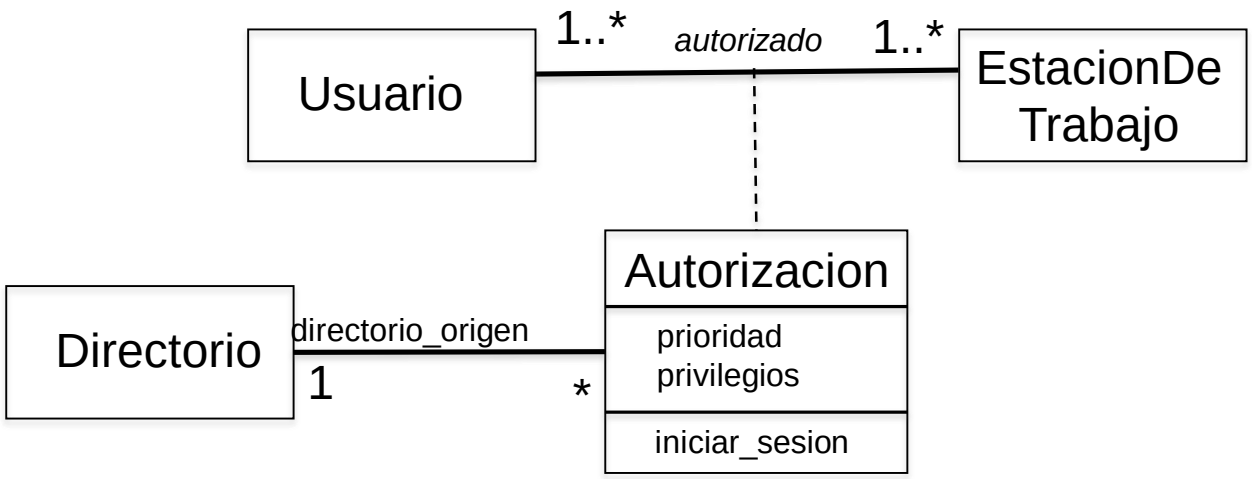
- También pueden aparecer en asociaciones con multiplicidad máxima 1



- En este caso pueden situarse en la parte muchos
- Pero, ¿qué pasa si la multiplicidad 1..1 es 0..1?

Clases asociación ...

- Como cualquier clase, una clase asociación puede tener métodos además de atributos

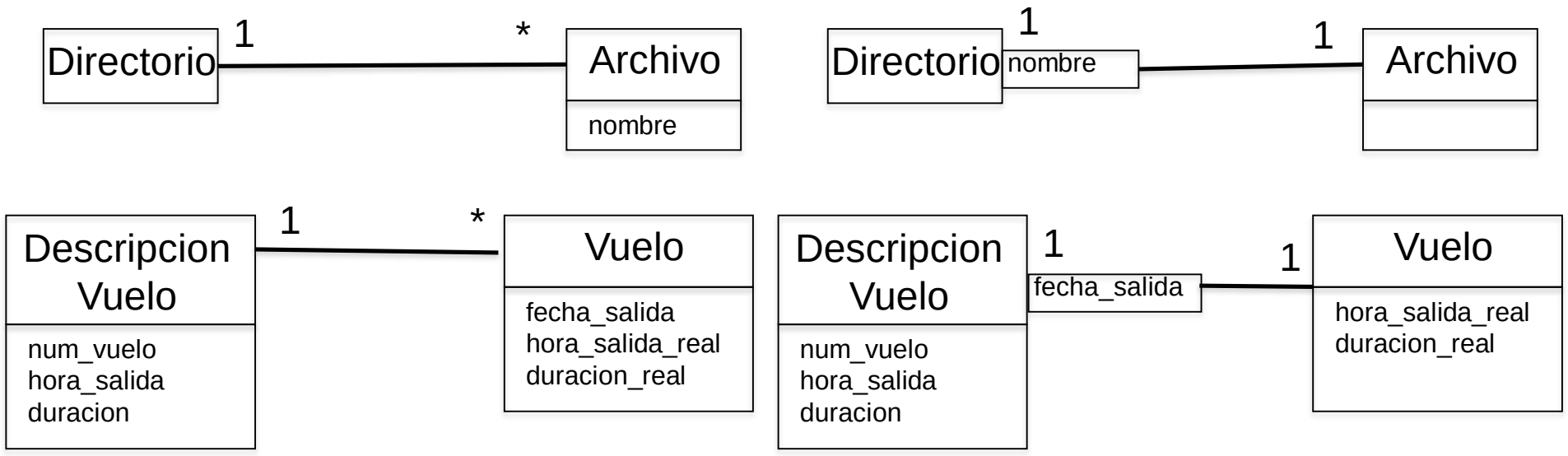


Calificación

- Problema de las búsquedas
- Dado un objeto en un extremo de la asociación ¿cómo identificar un objeto o conjunto de objetos en el otro extremo?
- Un calificador es un atributo de la asociación cuyos valores identifican un subconjunto de objetos (normalmente un único objeto)
- El calificador es un atributo especial que reduce la multiplicidad de la asociación
- Una asociación calificada es una forma de asociación ternaria

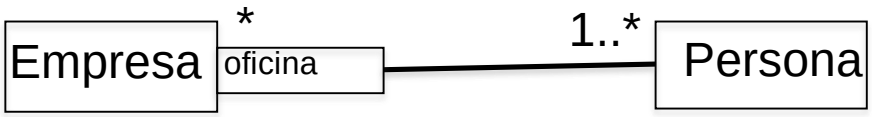
Calificación …

- La calificación mejora la precisión semántica del modelo e incrementa la visibilidad de las vías de navegación

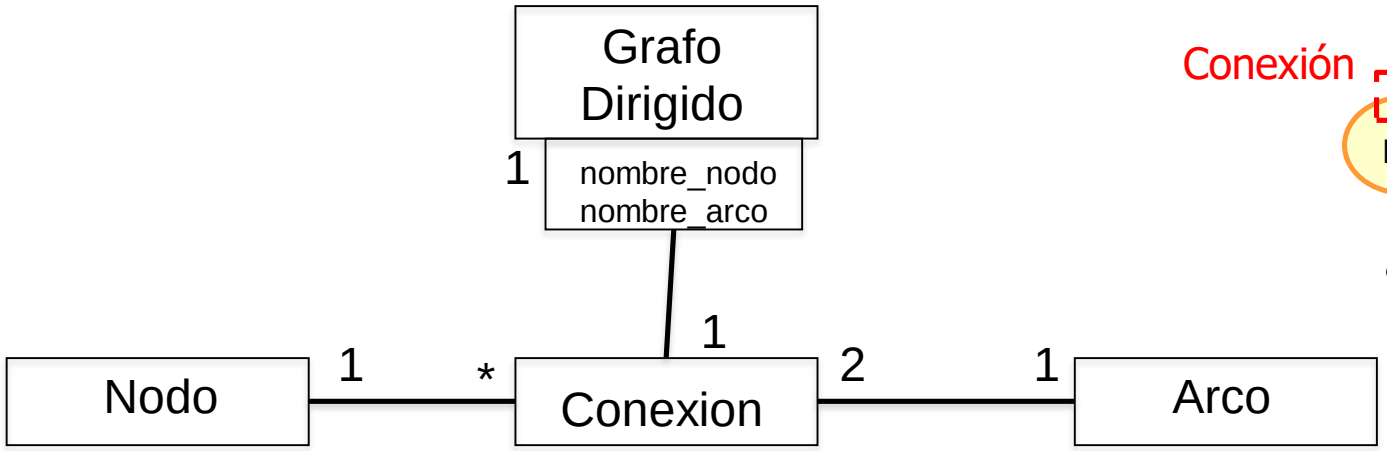


Calificación ...

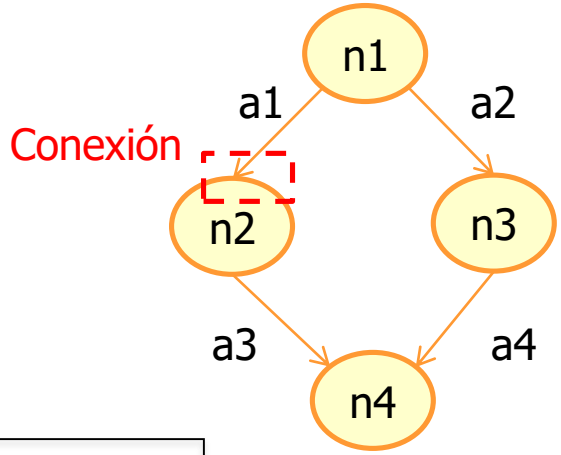
- En algunos casos no se reduce la multiplicidad



- Los calificadores pueden ser compuestos

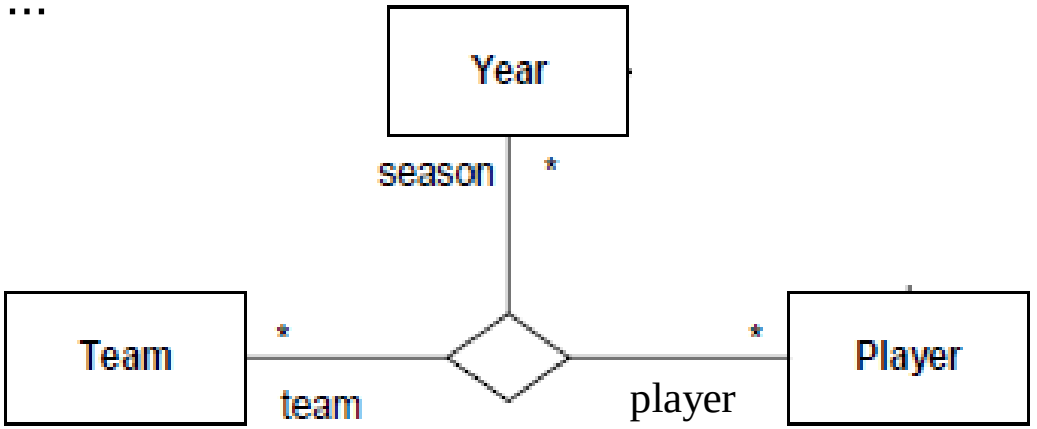


Ejemplo de grafo dirigido:



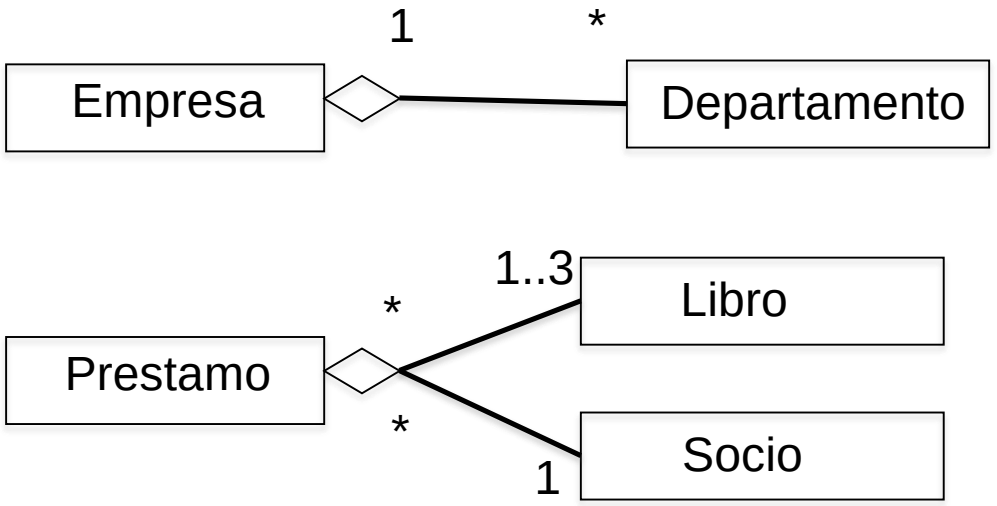
Asociaciones n-arias

- Cualquier asociación se puede dibujar como un diamante unido con una línea a cada extremo de la asociación
  - Las asociaciones con más de dos extremos solo se pueden dibujar de esta forma
- No son frecuentes:
  - Conforme avanza el análisis y el diseño se suelen descomponen o se reconvierten en asociaciones con calificadores, clases asociación, …



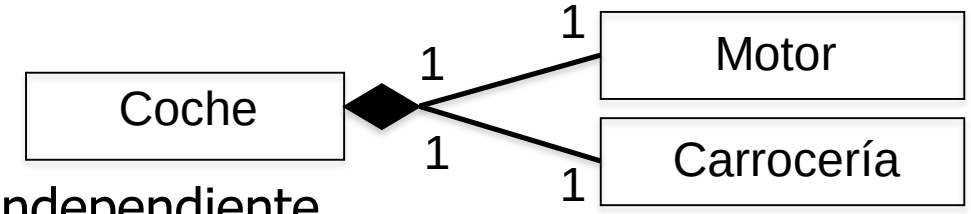
Agregación

- La agregación es una forma especial de asociación.
- Modela una relación "todo/parte" o "tiene un"
- Una clase (el "todo") consta de elementos más pequeños (las "partes")



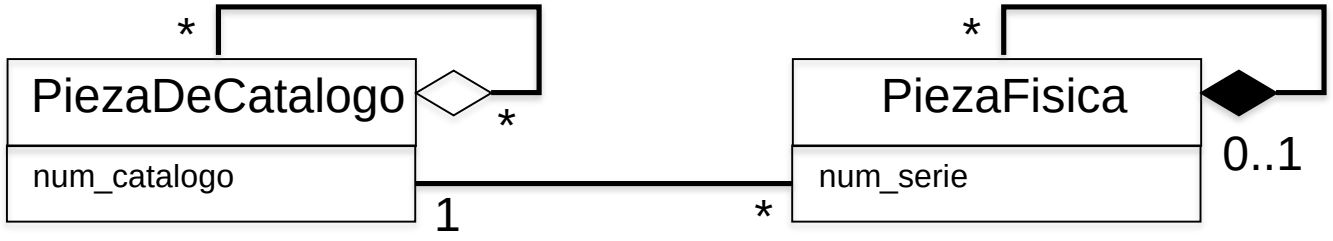
Composición

- La composición es una forma más restrictiva, más fuerte, de agregación
- La agregación no liga las vidas del todo y las partes
- La agregación sólo distingue el "todo" de las "partes"
- Composición:
  - Es una relación de pertenencia y vidas coincidentes de la "parte" con el "todo"
  - Las "partes" pueden crearse después del "todo" pero una vez creadas viven y mueren con él
  - El "todo" gestiona la creación y destrucción de las partes
  - Las partes no tienen existencia independiente



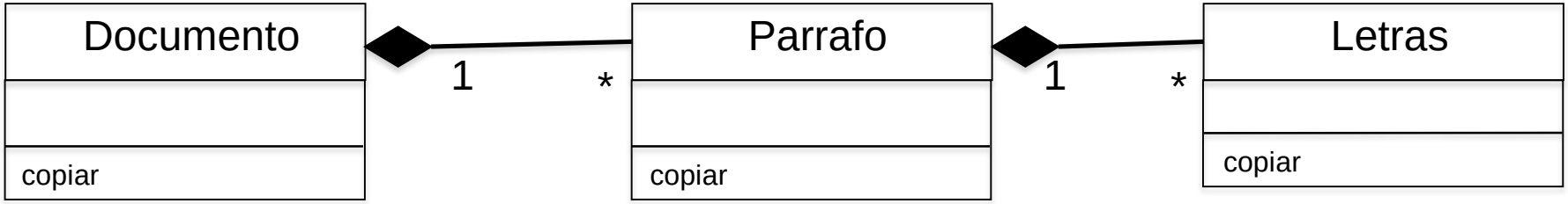
Composición ...

- En la composición, el "todo" también se llama "objeto compuesto" y las "partes" las "componentes"
- Diferencias agregación/composición:
  - En la composición, un objeto puede formar parte de sólo una parte compuesta a la vez. No se reutilizan los componentes
  - En la agregación, una "parte" se puede compartir por varios agregados



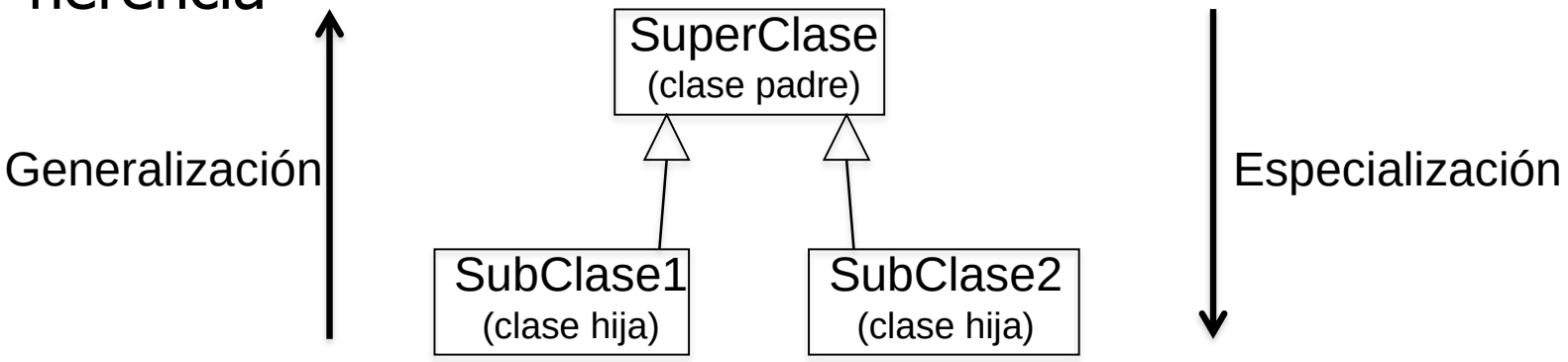
Composición ...

- Otra diferencia:
  - En la composición existe la propagación de operaciones y atributos



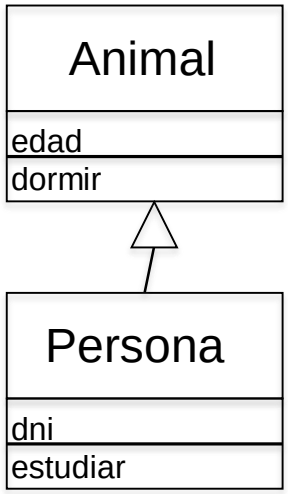
Herencia

- Desde el punto de vista de la implementación es un mecanismo de reutilización de código
- Desde un punto de vista conceptual es una abstracción para compartir similitudes entre clases al tiempo que se mantienen sus diferencias
- Especialización y generalización *son las dos caras* de la herencia



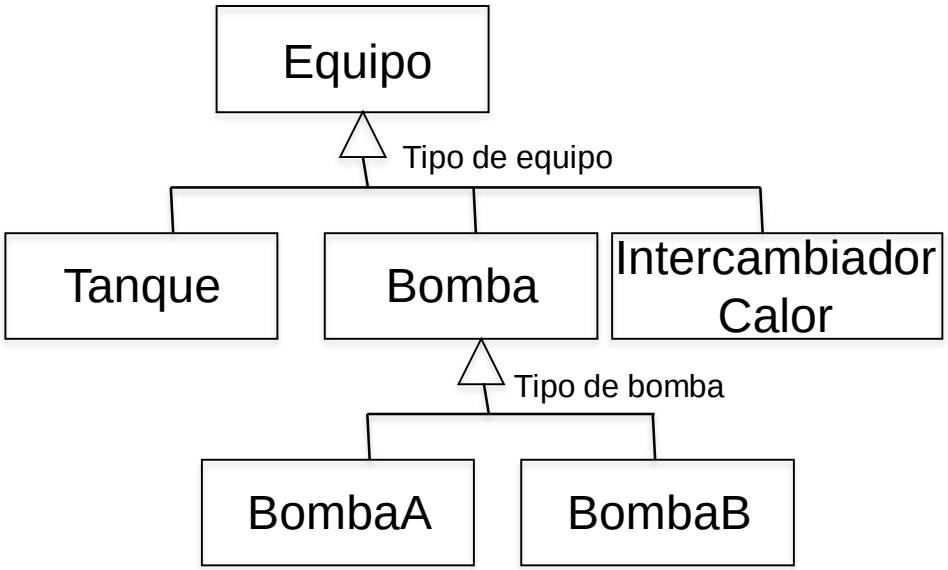
Herencia …

- También es conocida como relación *is-a* o *es-un*
- Cada instancia de una subclase es una instancia de la superclase
- Las subclases heredan atributos y métodos de la superclase, pudiendo añadir los suyos propios
  - Parte heredada o derivada
  - Parte emergente o incremental



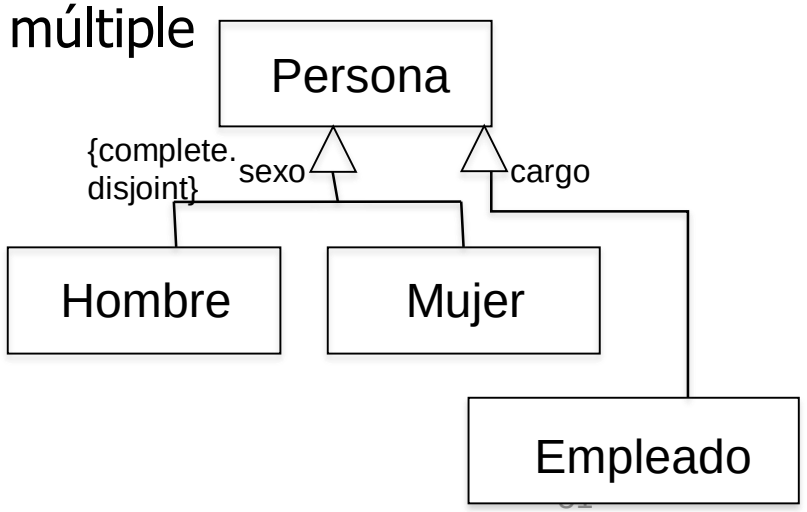
Herencia …

- Discriminadores
  - Indican qué propiedad de la generalización se abstrae



Herencia …

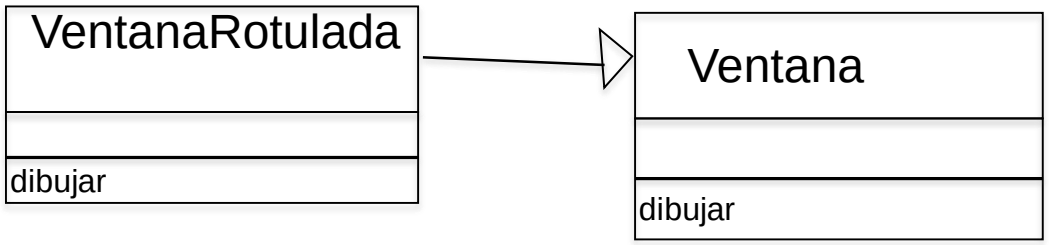
- Matices
  - *complete*: no se permiten clases hijas adicionales
  - *incomplete*: se permiten clases hijas adicionales
  - *disjoint*: los objetos de la clase padre no pueden tener más de una de las clases hijas como tipo; clases mutualmente incompatibles si se crea una nueva clase hija con herencia múltiple
  - *overlapping*: los objetos de la clase padre pueden tener más de una de las clases hijas como tipo; clases compatibles si se crea una nueva clase hija con herencia múltiple
  - Posibles combinaciones:
    {complete,disjoint}
    , {complete,overlapping}
    , {incomplete,disjoint}
    , {incomplete,overlapping}



Redefinición de operaciones

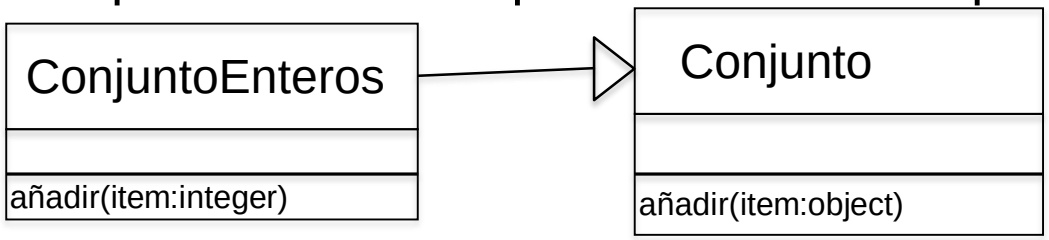
- Por definición de especialización, las operaciones aplicables a una clase lo son a todas sus subclases
- Los métodos heredados pueden ser redefinidos por los siguientes motivos:
  - Redefinición por extensión
    - La nueva operación es la misma que la heredada pero añade nuevo comportamiento que suele afectar a los nuevos atributos de la subclase

```
(implementación de dibujar)
begin
   hereda dibujar()
   {código emergente}
end
```

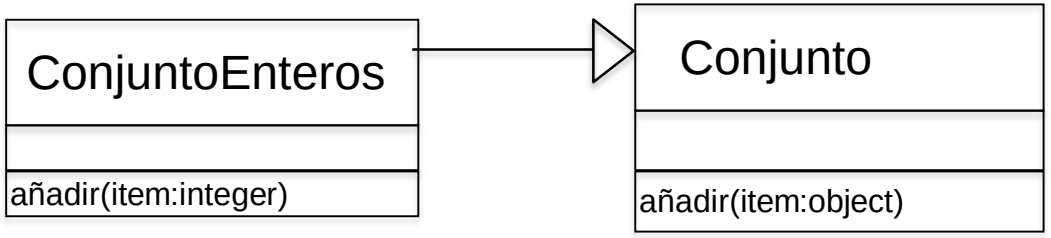


Redefinición de operaciones …

- Redefinición por restricción
  - La nueva operación restringe el protocolo de la heredada. La restricción no puede afectar al número de parámetros, únicamente al tipo. Los nuevos tipos deben ser subtipos de los iniciales



- Redefinición por optimización
  - Aprovecha las características de la nueva clase añadiendo un código más eficiente

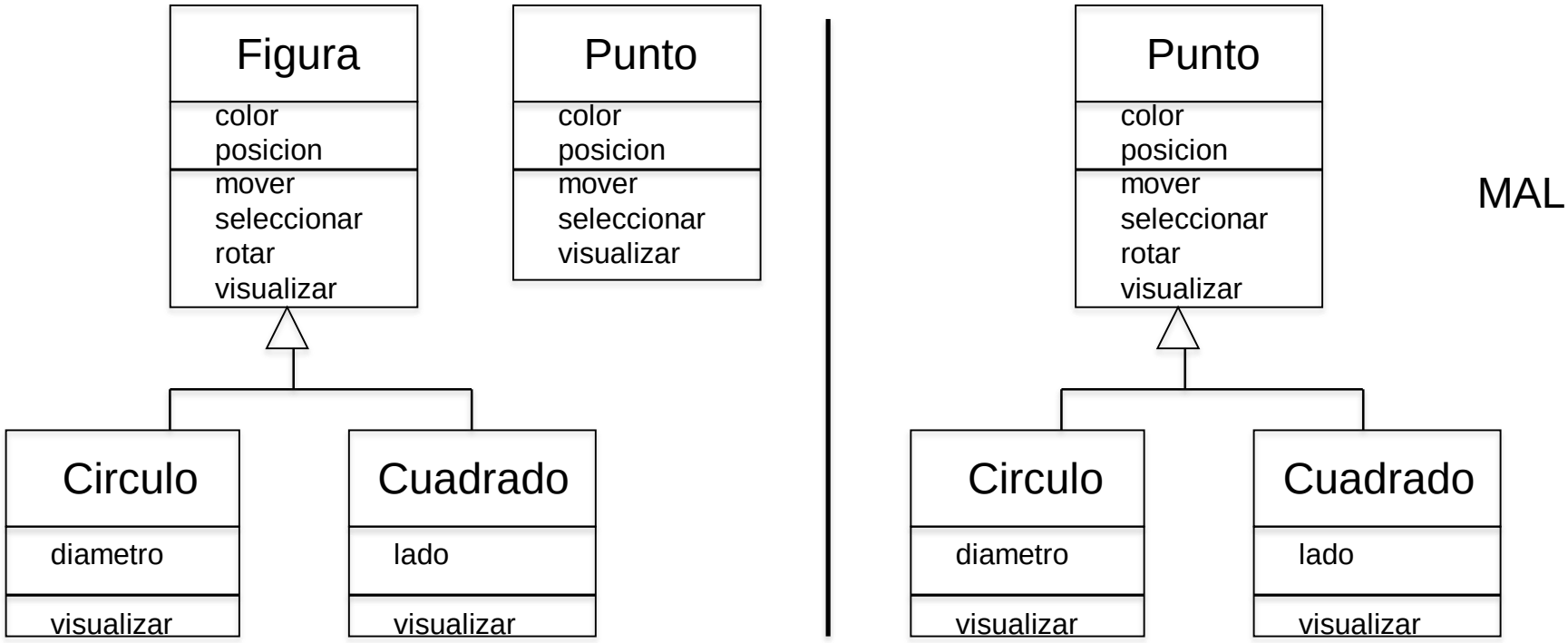


Redefinición de operaciones ...

- Consideraciones
  - Se heredan todas las operaciones: consulta, actualización, creación y destrucción
  - Una operación no puede ser redefinida para que se comporte de modo distinto
  - Se puede añadir un comportamiento adicional
  - Todas las operaciones deben tener la misma interfaz

Herencia ...

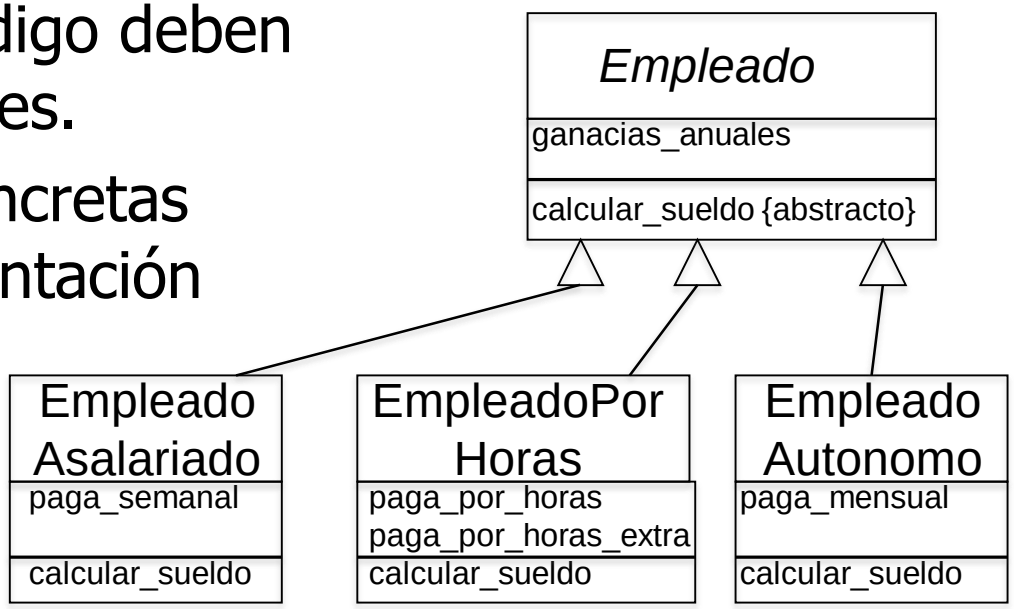
- En Análisis la herencia es conceptual no un mero mecanismo de reutilización de código



MAL

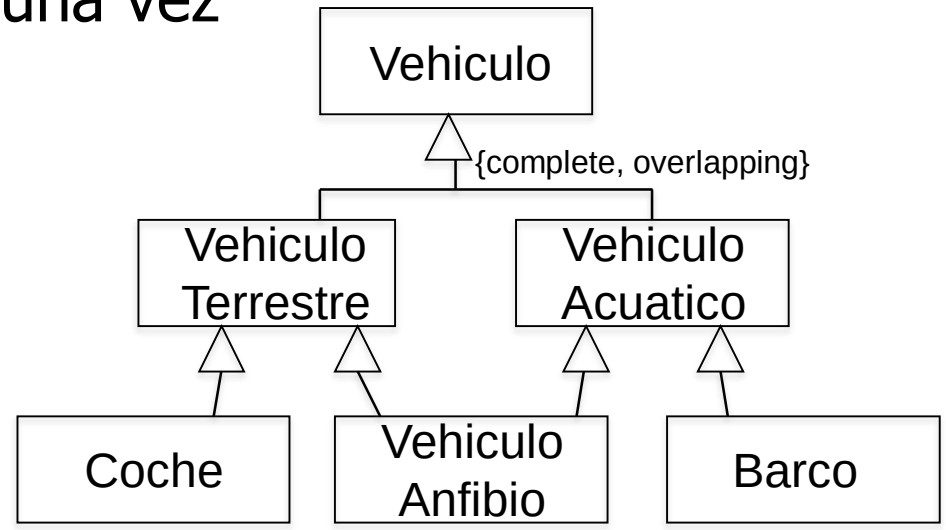
Clases abstractas

- Una clase abstracta no posee instancias, pero sus descendientes sí
- Una clase abstracta tiene al menos una operación sin código y por tanto no pueden instanciarse
- Las operaciones sin código deben definirse en las subclases.
- Todas las subclases concretas suministran la implementación
- Una clase abstracta puede implementarse en su totalidad (o parcialmente)



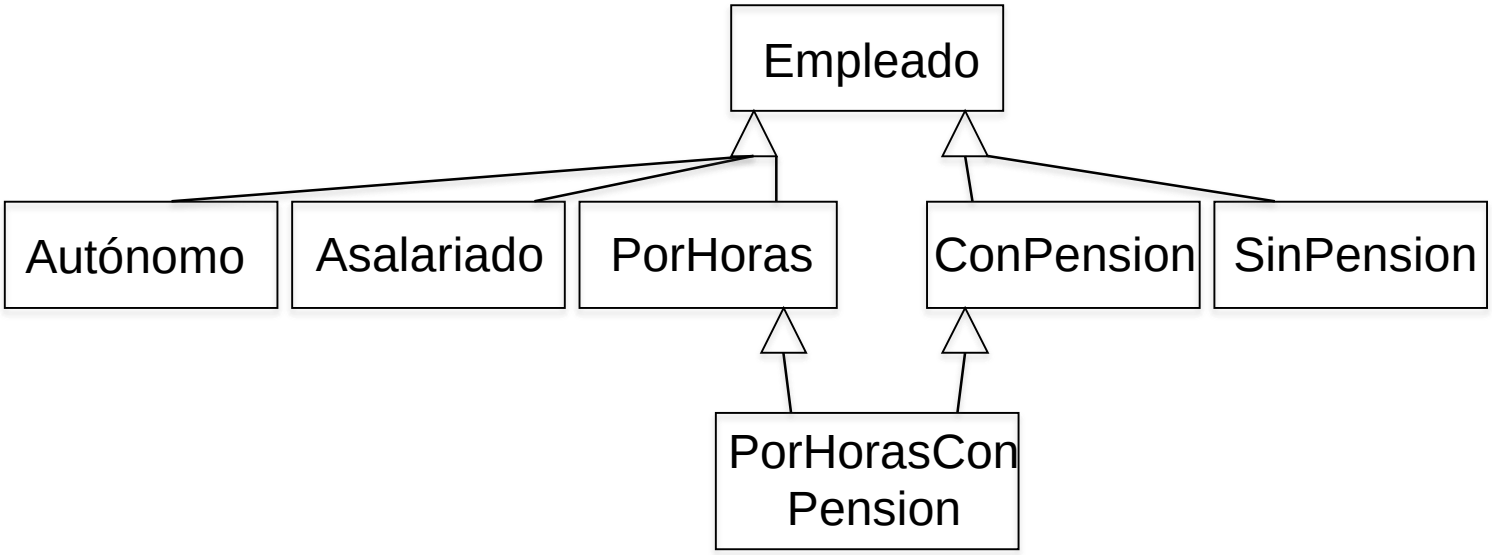
Herencia múltiple

- Cuando una clase tiene más de una clase antecesora directa
- Hereda los atributos y operaciones definidos en todas las clases antecesoras
- Una característica que llegue por distintas vías sólo se heredará una vez



Herencia múltiple …

- Resolver la carencia de herencia múltiple debería ser una cuestión de implementación, pero una reestructuración temprana del modelo puede ser una forma sencilla de evitar el problema

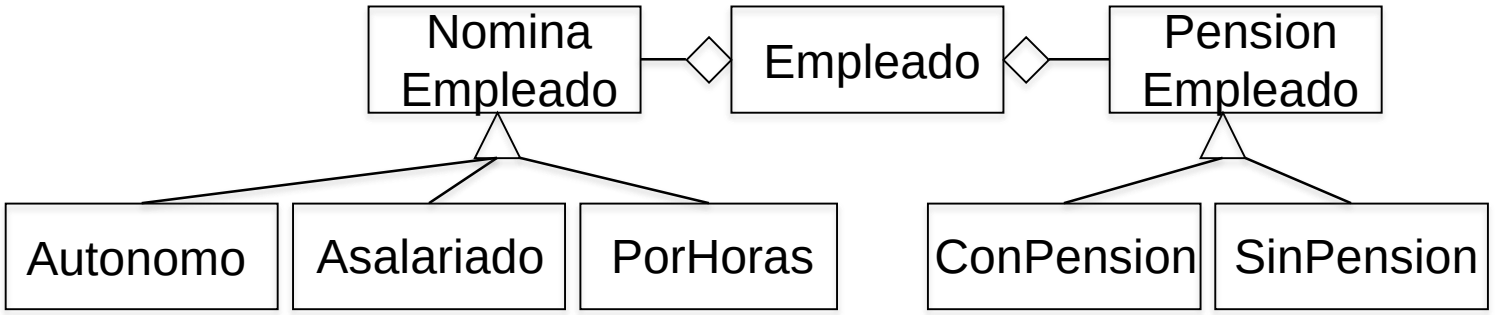


Delegación ...

- Mecanismo mediante el cual un objeto le pasa una operación a otro para que la ejecute

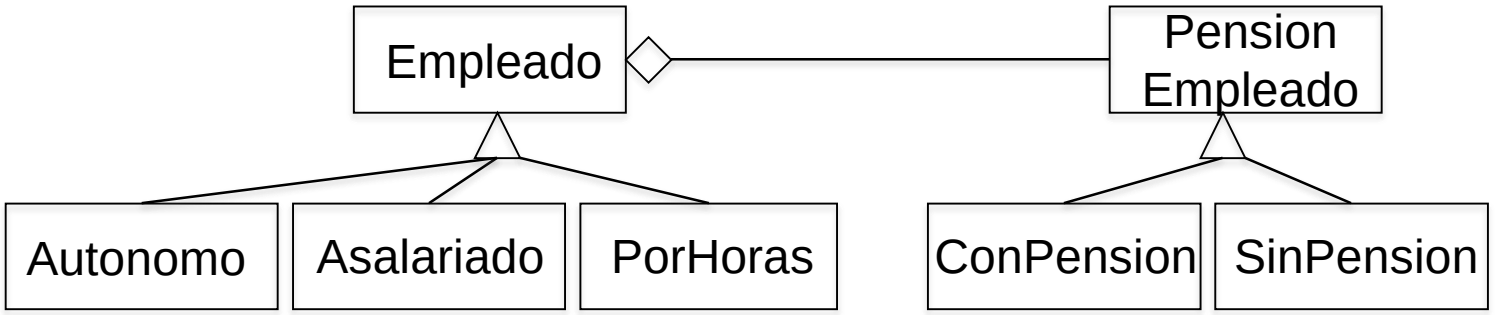
Reestructuración herencia múltiple

- Delegación empleando agregación de roles
  - La superclase se refunde en forma de agregado, donde cada componente sustituye a una generalización
  - Sustituye un único objeto (con una única identidad) por un grupo de objetos



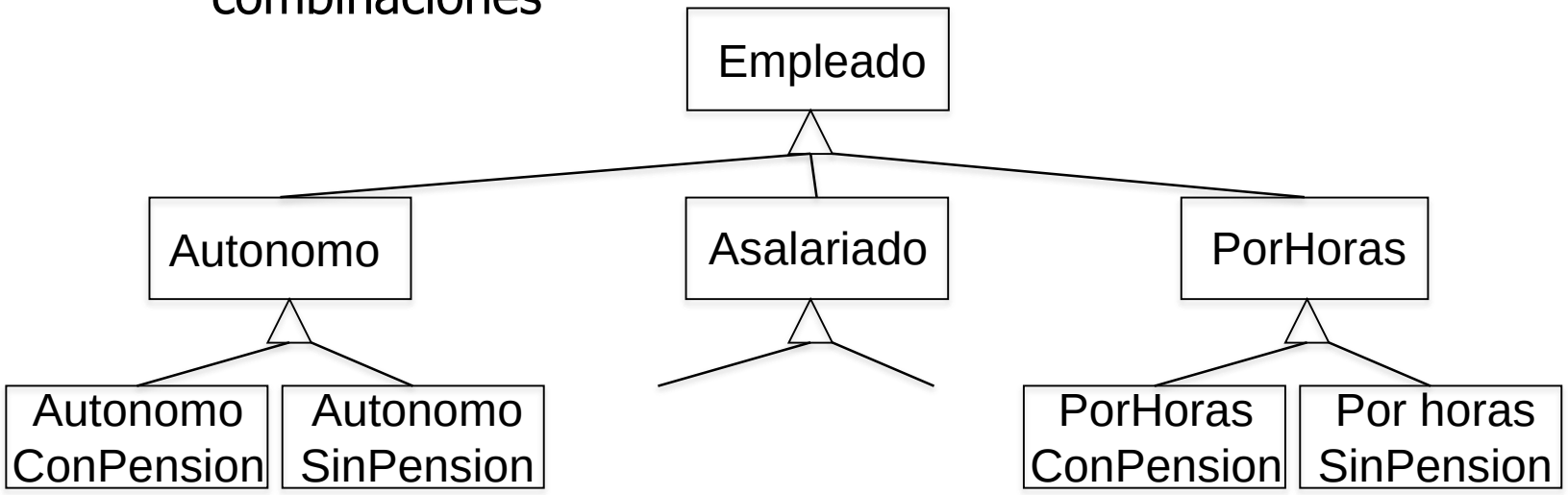
Reestructuración herencia múltiple

- Heredar la clase más importante y delegar el resto
  - Se mantiene la identidad y la herencia a través de una generalización



Reestructuración herencia múltiple

- Generalización anidada
  - Primero se factoriza por una generalización y después por otra
  - Mantiene la herencia pero se multiplican las posibles combinaciones



Reestructuración herencia múltiple

- Cualquiera de los trucos anteriores puede funcionar pero comprometen la lógica y el mantenimiento:
  - Si una subclase tiene varias superclases, todas ellas igualmente importantes, quizá sea mejor utilizar delegación
  - Si una superclase es más importante, quizá usar herencia simple y delegación
  - Si el número de combinaciones es pequeño, quizá anidar
  - Si una superclase tiene más atributos que las otras (o es cuello de botella para el rendimiento), mantener la herencia a través de esa vía
  - Si se decide anidar, hacerlo primero por el criterio más importante y así sucesivamente
  - Si es importante mantener la identidad, sólo la generalización anidada es capaz

4. Técnica de construcción del modelo de objetos

- La creación de un diagrama de clases es un proceso de abstracción y entendimiento del problema que requiere experiencia
  - Existen guías con pasos recomendados (ver a continuación)
  - También ayuda construir previamente un diagrama de objetos donde se pueda ver una instantánea con ejemplos de los objetos (instancias) que necesitamos representar en nuestro sistema
- El resultado será una entrada para la fase de Diseño que proporciona una descripción completa de los conceptos existentes en el dominio del problema

Pasos recomendados (I)

- Paso 1: Identificar clases
  - Hacer una lista inicial exhaustiva con los sustantivos que aparecen en los requisitos
- Paso 2: Retener las clases correctas
  - Revisar lista anterior para descartar:
    - Clases redundantes, irrelevantes (¿tiene que ver con el dominio del problema?) o poco precisas (una clase debe ser específica)
    - Atributos: los nombres que describen objetos individuales pueden ser considerados atributos
    - Operaciones: Si un nombre describe una operación que se aplica a objetos no lo consideramos una clase
    - Roles: El nombre de una clase refleja su naturaleza intrínseca y no el rol que juega en una asociación

Pasos recomendados (II)

- Paso 3: Preparar un diccionario de datos
  - Para cada clase se escribe un pequeño *párrafo* que la describe con precisión: alcance de la clase dentro del problema incluyendo todas las suposiciones y restricciones sobre sus miembros
  - Ejemplo: clase cuenta en un Cajero Automático
    - Cuenta: *Cuenta individual de un banco a la que se pueden aplicar transacciones. Las cuentas pueden ser de varios tipos; como mínimo serán de ahorro o corrientes. Un cliente puede tener más de una.*

Pasos recomendados (III)

- Paso 4: Identificar asociaciones, agregaciones y composiciones
  - Suelen corresponderse con verbos en el dominio del problema
  - Hacen referencia a localizaciones físicas, relaciones de pertenencia o posesión, verificación de alguna condición (trabaja para, dirige, casado con, …), …
- Paso 5: Retener las asociaciones, … correctas
  - Revisar lista anterior para descartar:
    - Asociaciones irrelevantes o de implementación
    - Acciones
    - Requisitos expresados como acciones
    - Asociaciones ternarias
    - Asociaciones derivadas

Pasos recomendados (IV)

- Paso 6: Identificar atributos
  - No suelen aparecer en la definición del problema (en los requisitos)
  - Descartar:
    - Identificadores
    - Atributos discordantes
  - Intentar plantear modelos simples
    - calificadores y clases asociación solo si son relevantes, …

Pasos recomendados (V)

- Paso 7: Organizar el diagrama utilizando generalización/especialización (herencia)
  - Estudiar de nuevo el modelo aplicando generalización/especialización
- Paso 8: Iterar el modelo de objetos
  - El análisis es un proceso iterativo (primero se construye un subconjunto y luego se va extendiendo)
  - ¿Faltan clases?
  - ¿Sobran clases?
  - ¿Faltan o sobran asociaciones?
- Paso 9: Agrupar clases en subsistemas (… continuará en el tema de Diseño del Sistema)