TEMA 3.3:DIAGRAMA DE CLASES

Índice

- Introducción
- Elementos del diagrama de clases
- Conceptos básicos de orientación a objetos
- Clases
- Conectores
  - Dependencia
  - Asociación
  - Generalización/Especialización
  - Agregación y Composición
  - Interfaces
- Ejercicios

Introducción

- Los **diagramas de clases** se caracterizan por ser **estáticos**
  - No describen acciones
  - Muestran entidades y sus relaciones

Introducción

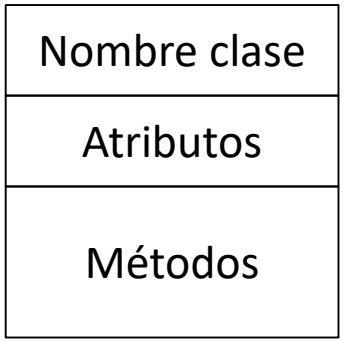
- Si pensamos en las cosas que nos rodean, muchas de ellas tienen atributos (propiedades) y pueden realizar determinadas acciones o tareas
- Todas las cosas las podemos clasificar en categorías (mobiliario, telefonía, automóviles, etc)
- A estas **categorías** se les llamará **clases**
- Una clase es una categoría o grupo de cosas que tienen **atributos y acciones similares**

Introducción

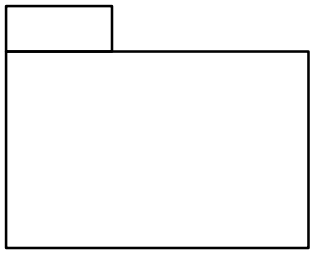
- El diagrama de clases captura el **vocabulario del sistema**
- Este diagrama se crea en las primeras fases de modelado y se va refinando a lo largo de todo el proceso de desarrollo
- El principal propósito del Diagrama de Clases se puede resumir en tres puntos:
  - Nombrar y modelar conceptos del sistema.
  - Especificar colaboraciones.
  - Especificar esquemas lógicos de bases de datos.

Elementos del diagrama de clases

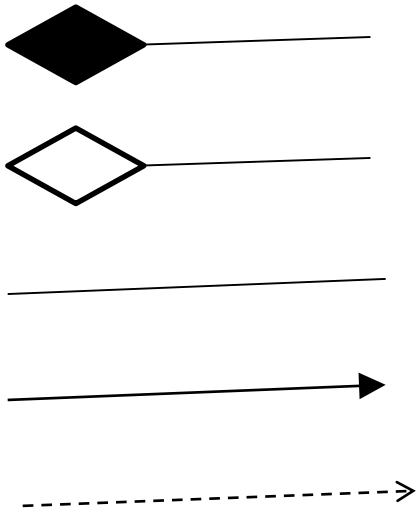
- Clases



- Paquetes



- Conectores



- Multiplicidad

, 0.., 1..*, M,N, 1

- Notas

Conceptos básicos de OO

- Abstracción
- Herencia
- Polimorfismo
- Encapsulación
- Envío de mensajes

Conceptos básicos de OO

- Abstracción
  - La abstracción hace referencia a quitar ciertas propiedades y acciones de un objeto para dejar sólo aquellas que sean necesarias
  - Dependiendo del problema a resolver se requerirá distinto tipo de información, por ello, se podría prescindir de ciertas propiedades o acciones

Conceptos básicos de OO

- Herencia
  - Un objeto es una instancia de una clase
  - Como instancia de una clase, un objeto tiene todas las características de la clase de la que proviene
  - Esta propiedad se denomina herencia
  - No importa si el objeto no utiliza todas las características de la clase, los heredará todos
  - Una clase puede heredar de otra clase

Conceptos básicos de OO

- Envío de mensajes
  - El único mecanismo para modificar el estado actual de un objeto son sus METODOS o SERVICIOS
  - Los objetos se comunican entre sí mediante el paso de mensajes
  - El envío de un mensaje consiste en indicar a un objeto que ejecute un servicio
    - ObjetoDestino.Servicio(parámetros)

Conceptos básicos de OO

- Polimorfismo
  - Una operación puede tener el mismo nombre en diferentes clases
  - Por ejemplo, la función abrir() se puede asociar a: una ventana, una puerta, una cuenta bancaria, etc
  - Cada clase “sabe” cómo realizar esa operación
  - El polimorfismo permite al modelador mantener la terminología sin tener que crear palabras artificiales para evitar la duplicación

Conceptos básicos de OO

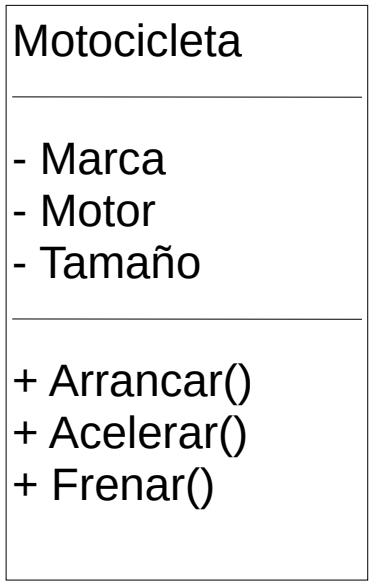
- Encapsulación
  - La encapsulación está relacionada con la ocultación de información
  - Se trata de agrupar bajo una misma entidad los datos y las funciones (métodos) que trabajan con esos datos
  - Se conocen las funcionalidades pero no cómo operan internamente
  - El objetivo es independizar la implementación interna del interfaz del objeto

Clases

- Una clase determina el ámbito de definición de un conjunto de objetos
- Cada objeto pertenece a una clase
- Los objetos se crean por instanciación de las clases

Clases

- Cada clase se representa como un rectángulo con tres apartados:
  - Nombre de la clase
    - Estereotipos (<< estereotipo>>)
  - Atributos de la clase
    - Modificadores de acceso (+ - # ~)
  - Métodos de la clase
    - Modificadores de acceso (+ - # ~)



Clases: Estereotipos

- Un **estereotipo** es un **nuevo tipo de elemento** de modelado que **extiende** la semántica del metamodelo
- UML proporciona varios estereotipos predefinidos
- UML permite añadir nuevos estereotipos si es necesario
  - Para extender, alterar el significado, características o sintaxis de un elemento de modelado de UML
    - <<Interface>>
    - <<Metaclass>>

    http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/
    (ANEXO C)

Atributos

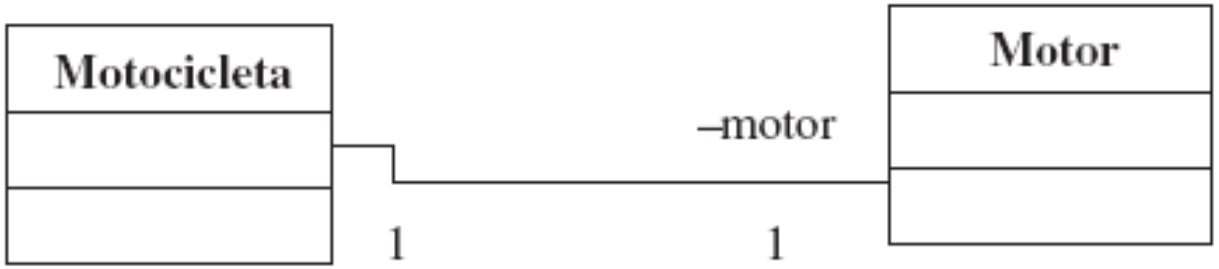
- Un atributo es una propiedad o característica de una clase que se declara mediante un nombre
  - Se les asigna un modificador de acceso para determinar la visibilidad
- Un atributo describe un rango de valores que podrán tomar los objetos
- Todo objeto de una clase tiene un valor específico para cada atributo
- UML permite añadir información adicional a los atributos, como por ejemplo, el tipo del atributo, un valor predeterminado, etc

Atributos

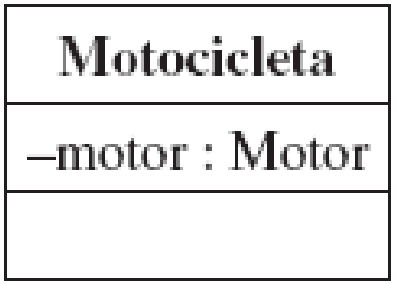
- Ejemplo:
  - Atributos para una Motocicleta

- Motor: TipodeMotor=TipodeMotor.DosTiempos
- Tamaño: cadena="220cc"
- Marca : cadena="Kawasaki" {sólo lectura}

Atributos



- Son equivalentes



Métodos

- Los métodos se muestran en el último rectángulo de la clase
- Un método es algo que la clase puede realizar o que otras clases pueden hacer con ella
- Los métodos tienen un modificador de visibilidad como los atributos
- En los métodos también es posible incluir información adicional
  - Los parámetros con sus tipos de datos o el tipo de valor devuelto

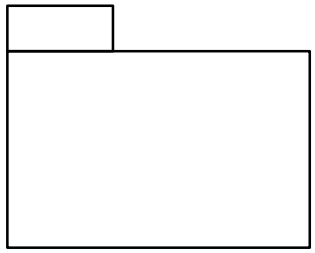
Encapsulación

- Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos
- La encapsulación presenta tres ventajas básicas:
  - Se protegen los datos de accesos indebidos.
  - El acoplamiento entre las clases se disminuye.
  - Favorece la modularidad y el mantenimiento.

Encapsulación

- UML tiene 4 niveles de visibilidad:
  - **Public (+):** Cualquier clase externa puede utilizar la característica (opción por defecto)
  - **Protected (#):** Sólo los descendientes de la clase pueden usarla
  - **Private (-):** Sólo la propia clase puede utilizarla
  - **Package (~):** Sólo las clases declaradas en el mismo paquete pueden utilizarla

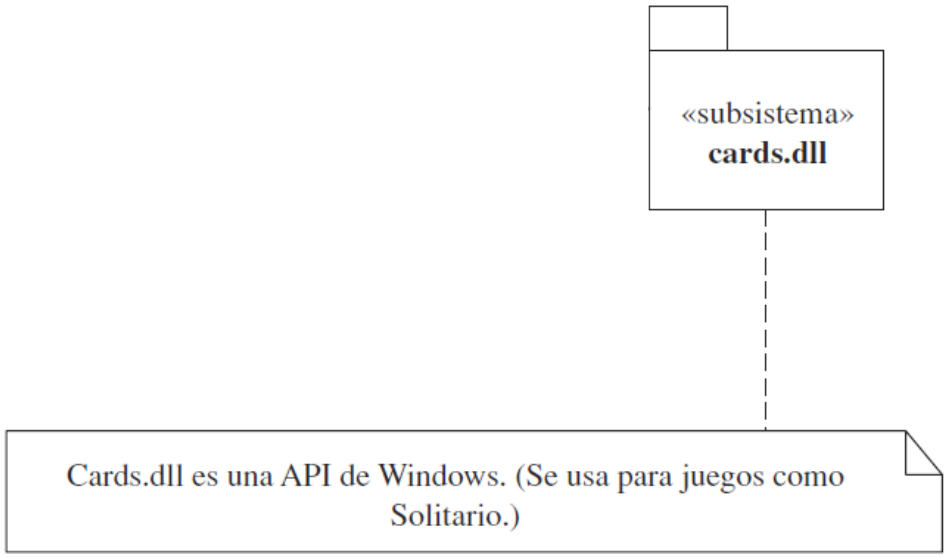
Paquetes



- El símbolo del paquete tiene la apariencia de una carpeta de archivos
- Se utiliza para representar un nivel más alto de abstracción que el de una clase
- Un paquete se puede implementar como un espacio de nombres o un subsistema
- También se puede usar para la organización de las clases y para representar una carpeta de archivos

Notas

- Los diagramas de clases permiten el uso de la nota
- Aunque es recomendable transmitir el significado de algo mediante las clases y sus relaciones evitando agregar demasiadas notas

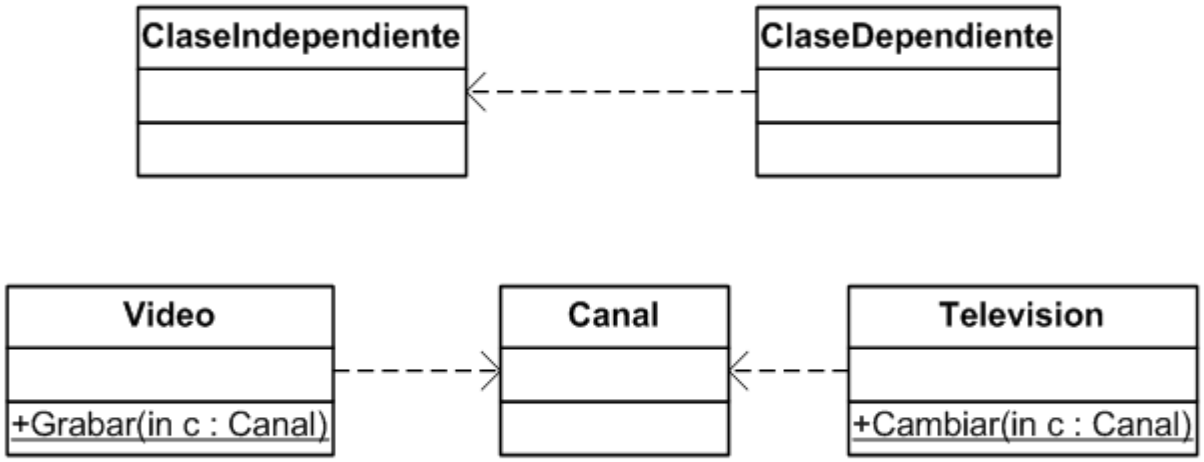


Conectores

- Los enlaces entre objetos pueden representarse entre las respectivas clases
- Formas de relación entre clases:
  - Dependencia
  - Asociación
    - Asociación de agregación
    - Asociación de composición
  - Generalización/Especialización
  - Interfaces

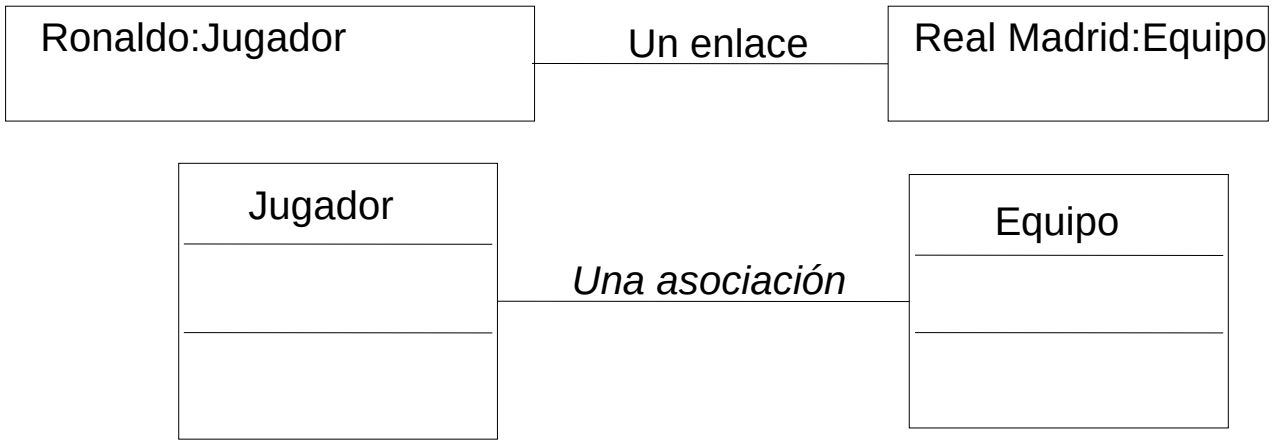
Conectores: Dependencia

- La dependencia es una relación semántica entre dos elementos en la cual un cambio en un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente)



Conectores: Asociación

- La asociación expresa una conexión bidireccional entre clases
- Se representa con una línea continua entre dos clases
- Una asociación es una abstracción de la relación existente en los enlaces entre los objetos

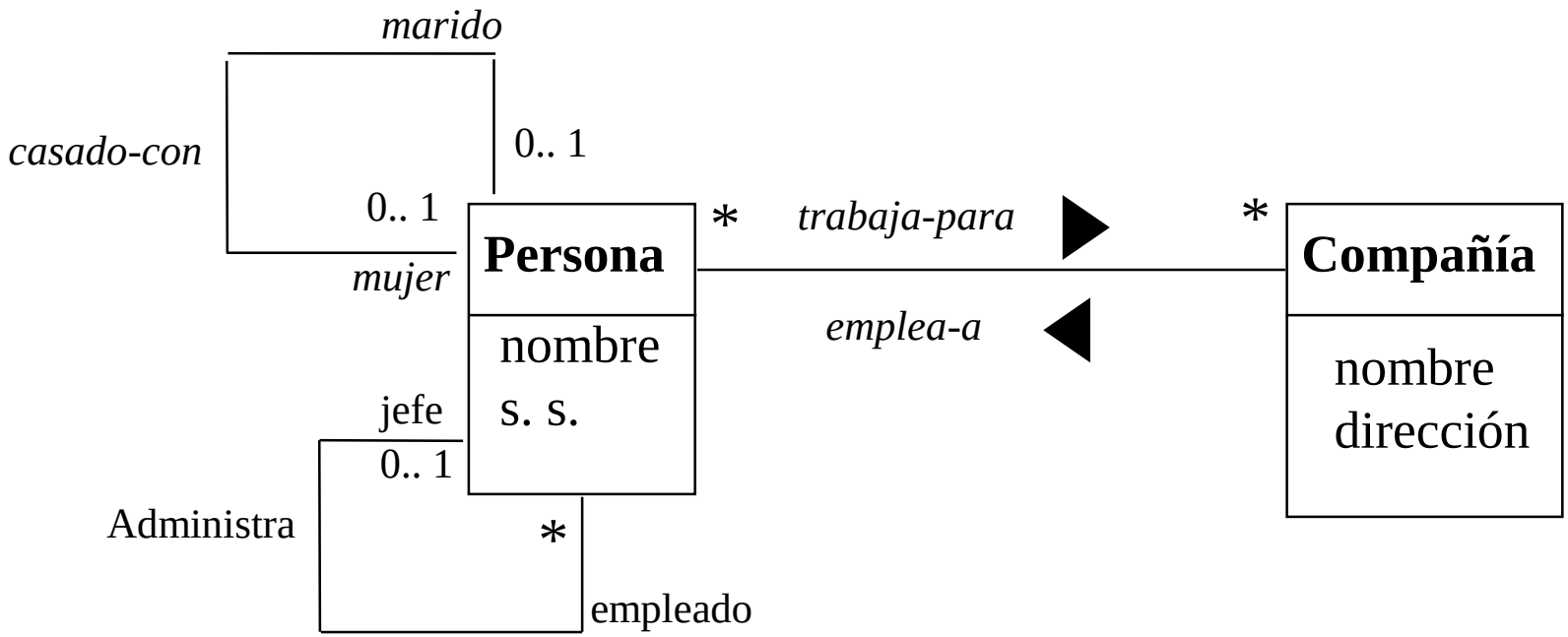


Conectores: Asociación

- Cada asociación se puede caracterizar:
  - "participa en", "emplea a", etc.
- La asociación se visualiza con una línea que une ambas clases con el nombre de la asociación justo en la línea.
  - Es útil indicar la dirección de la relación. Se utiliza un triángulo relleno que apunta en la dirección apropiada
- Cada clase asociada juega un papel (rol) dentro de tal asociación.
  - El rol se escribe sobre la línea de la asociación al lado de la clase que juega ese rol
    - Si pensamos en las clases Jugador y Equipo (profesional). En la asociación "participa en" el Jugador tiene el papel de "Empleado" y el Equipo de "Empleador".

Conectores: Asociación

- Ejemplo:

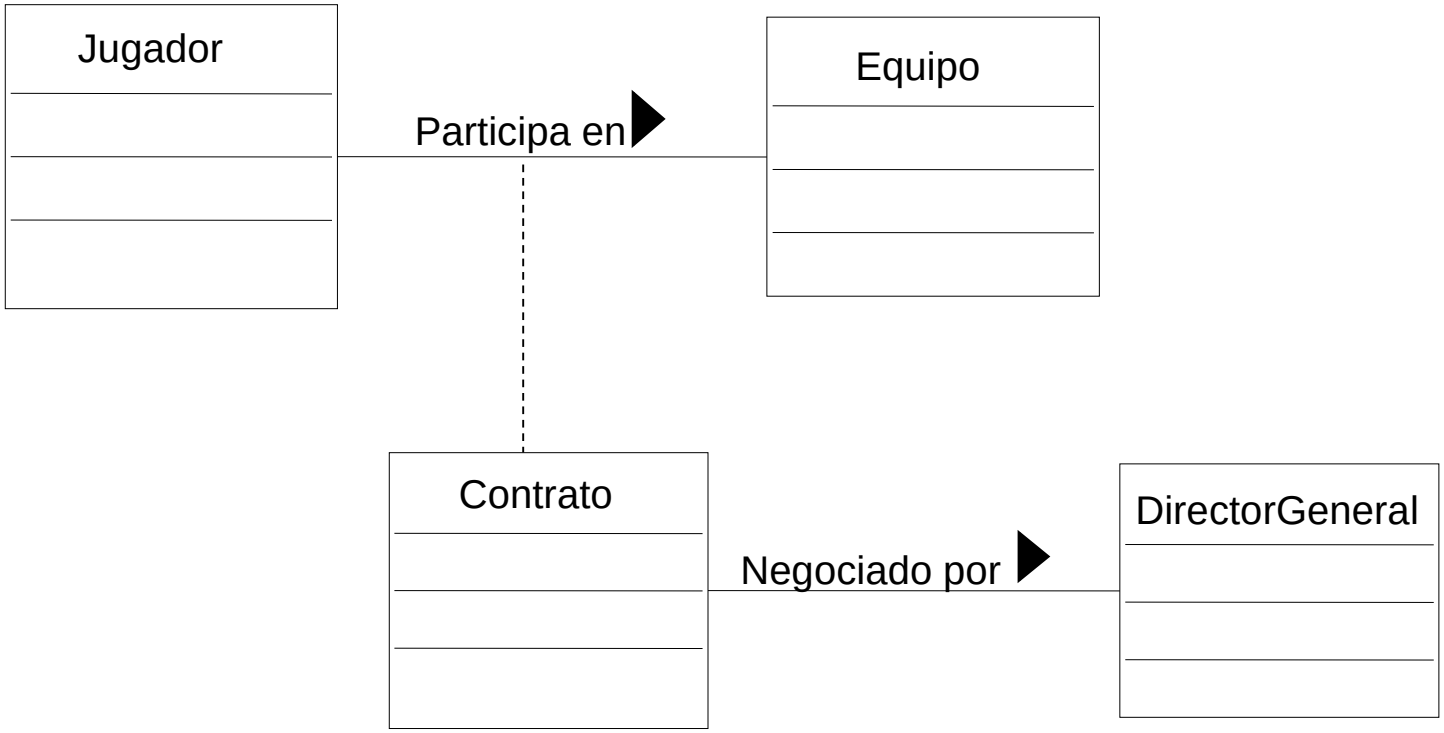


Conectores: Asociación

- Una asociación, al igual que las clases, puede tener atributos y operaciones:
  - En este caso se tiene una clase *Clase de Asociación.*
  - Se concibe de la misma manera que una clase estándar.
- Se utiliza una línea discontinua para conectarla a la línea de asociación
  - Sólo puede existir una instancia de la asociación entre cualquier par de objetos participantes

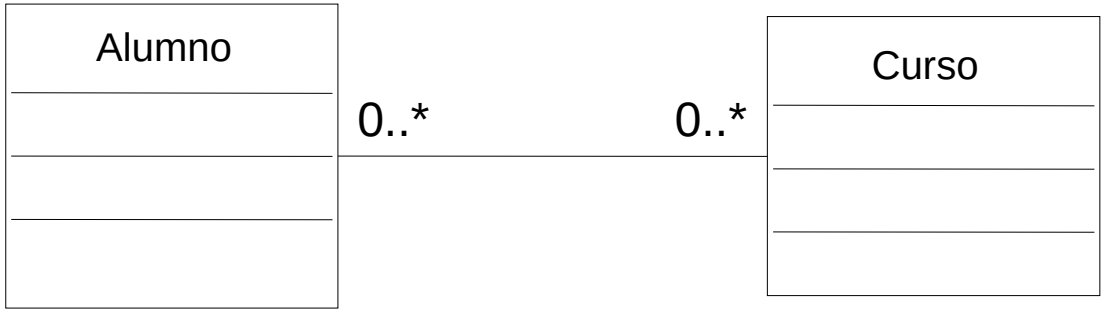
Conectores: Asociación

- Clase de asociación:



Conectores: Asociación

- Ejercicio:
  - Queremos llevar un historial de las calificaciones de todos nuestros alumnos
  - Existe una relación muchos a muchos entre la clase Alumno y la clase Curso

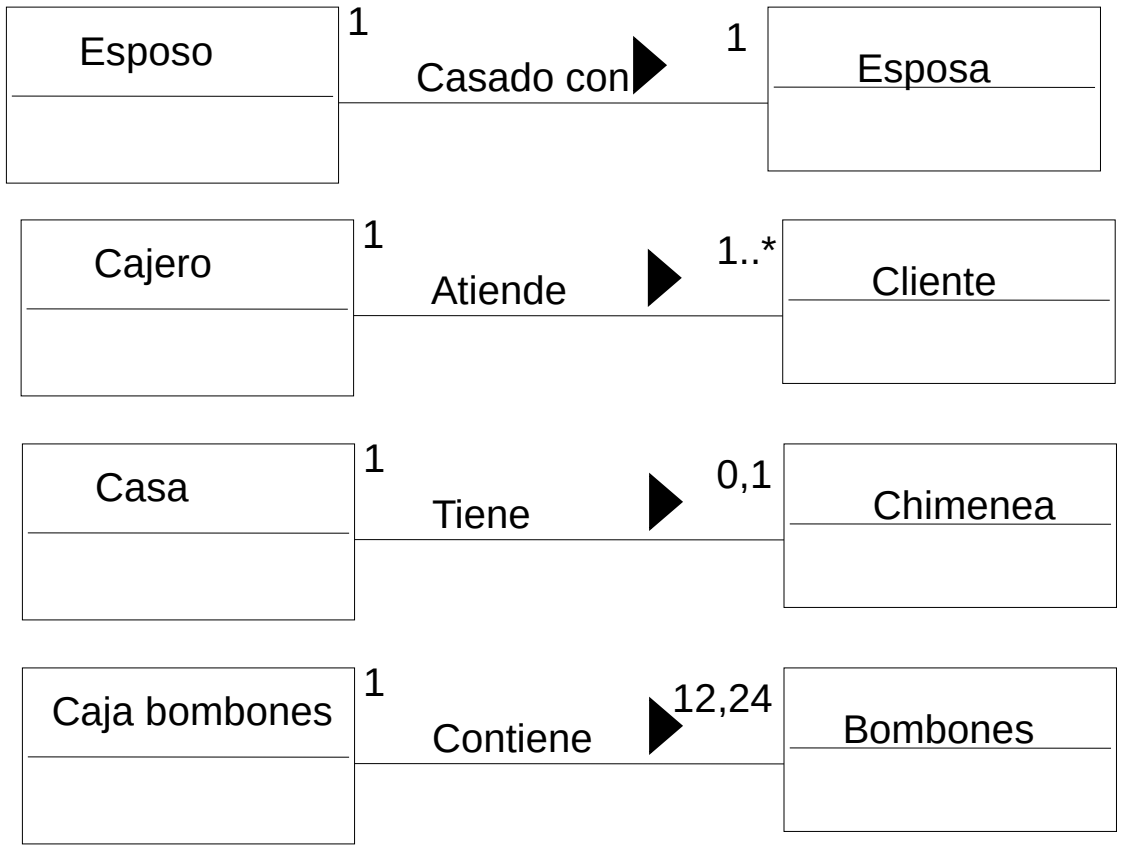


  - ¿Dónde pondremos los atributos de las calificaciones?

Conectores: Multiplicidad

- La multiplicidad se define como la cantidad de objetos de una clase que se relacionan con un objeto de la clase asociada
- Especificación de multiplicidad (mínima...máxima).
  - 1 Uno y sólo uno
  - 0..1 Cero o uno
  - M..N Desde M hasta N (enteros naturales)
  - * Cero o muchos
  - 0..* Cero o muchos
  - 1..* Uno o muchos (al menos uno)
  - La multiplicidad mínima >= 1 establece una restricción de existencia.

Conectores: Multiplicidad



Uno a uno

Uno a uno o más

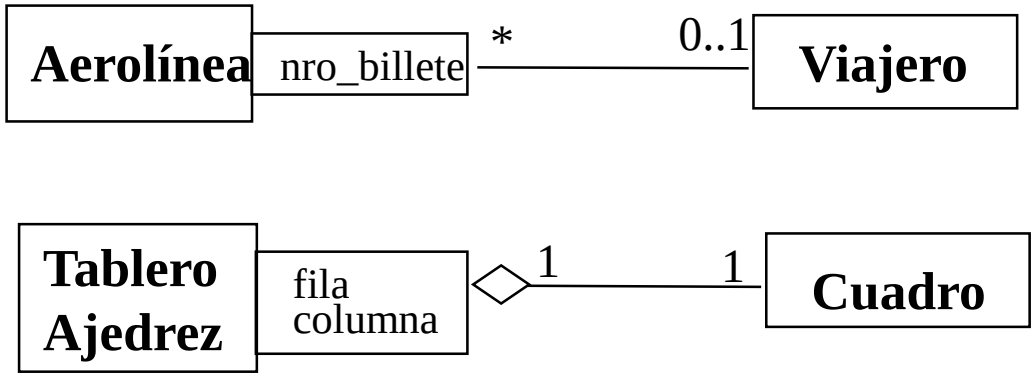
Uno a ninguno o uno

Uno a 12 o 24

Conectores: Asociación calificada

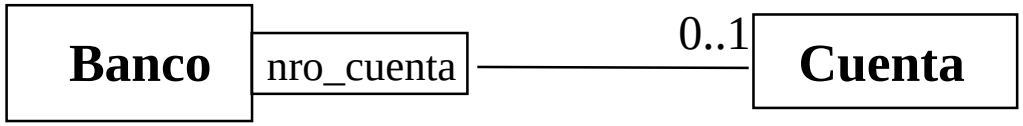
- Si la asociación es de 1 a muchos.
- Se presenta con frecuencia un reto: la búsqueda.
- Si un objeto de una clase tiene que seleccionar un objeto particular de otro tipo para cumplir con un papel en la asociación. La primera clase debe atenerse a un atributo para localizar al objeto adecuado.
- Dicho atributo es un identificador que puede ser un número de identidad.
  - Por ejemplo, cuando se realiza la reserva de un hotel, el hotel le asigna un número de confirmación
- En UML la información de identidad se conoce como calificador

Conectores: Asociación calificada

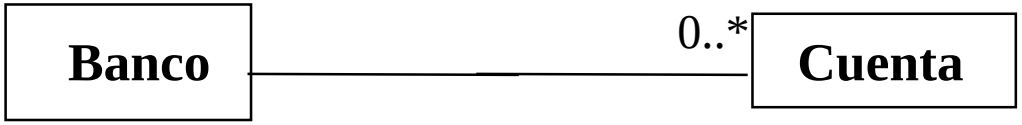


Conectores: Asociación calificada

- Ejemplo:



Asociación calificada

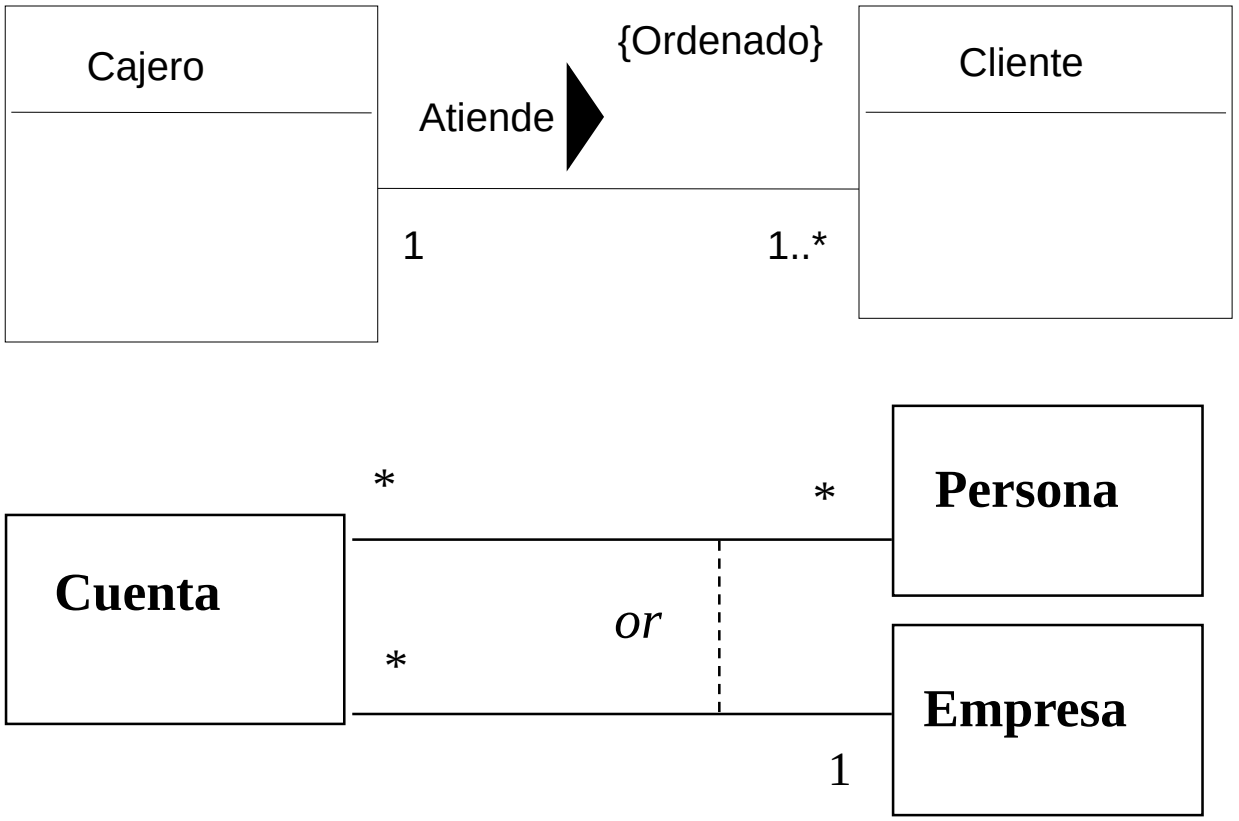


Asociación no calificada

Conectores: Restricciones

- Cuando una asociación entre dos clases debe seguir ciertas reglas
  - Necesita restricciones.
- Se indica con la restricción junto a la línea de la asociación.
- Ejemplo: un cajero atiende a un cliente, pero estos son atendidos en el orden de llegada, es decir en el orden en que están en la cola.
- Otro tipo de restricción es la Asociación excluyente.
  - Conocida como {or}. Por ejemplo una cuenta puede ser de una empresa o una persona.

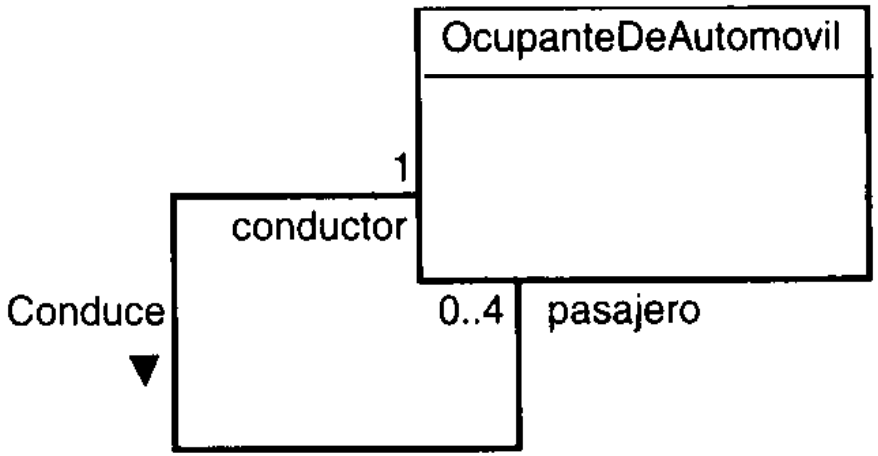
Conectores: Restricciones



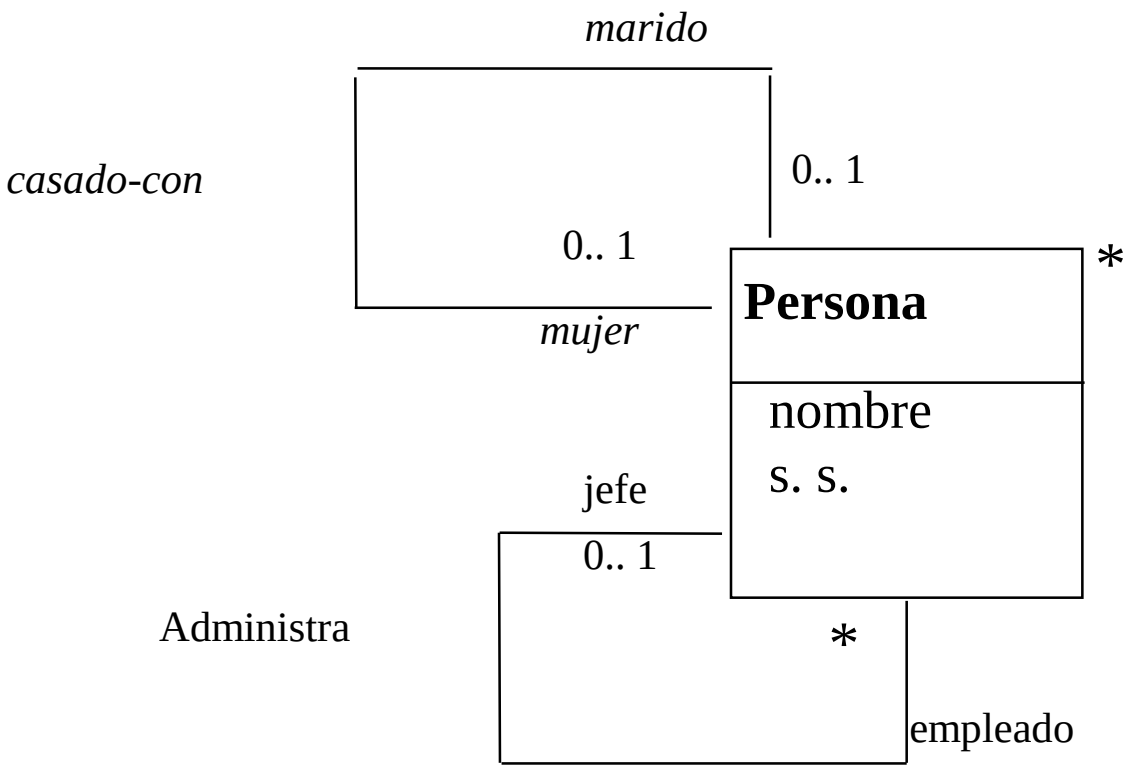
Asociación excluyente

Conectores: Asociación reflexiva

- A veces, una clase es una asociación consigo misma.
  - Ocurre cuando una clase tiene objetos que pueden jugar diversos papeles.
  - Un ocupante de un automóvil puede ser:
    - Conductor: Puede llevar ninguno o más ocupantes
    - Pasajero



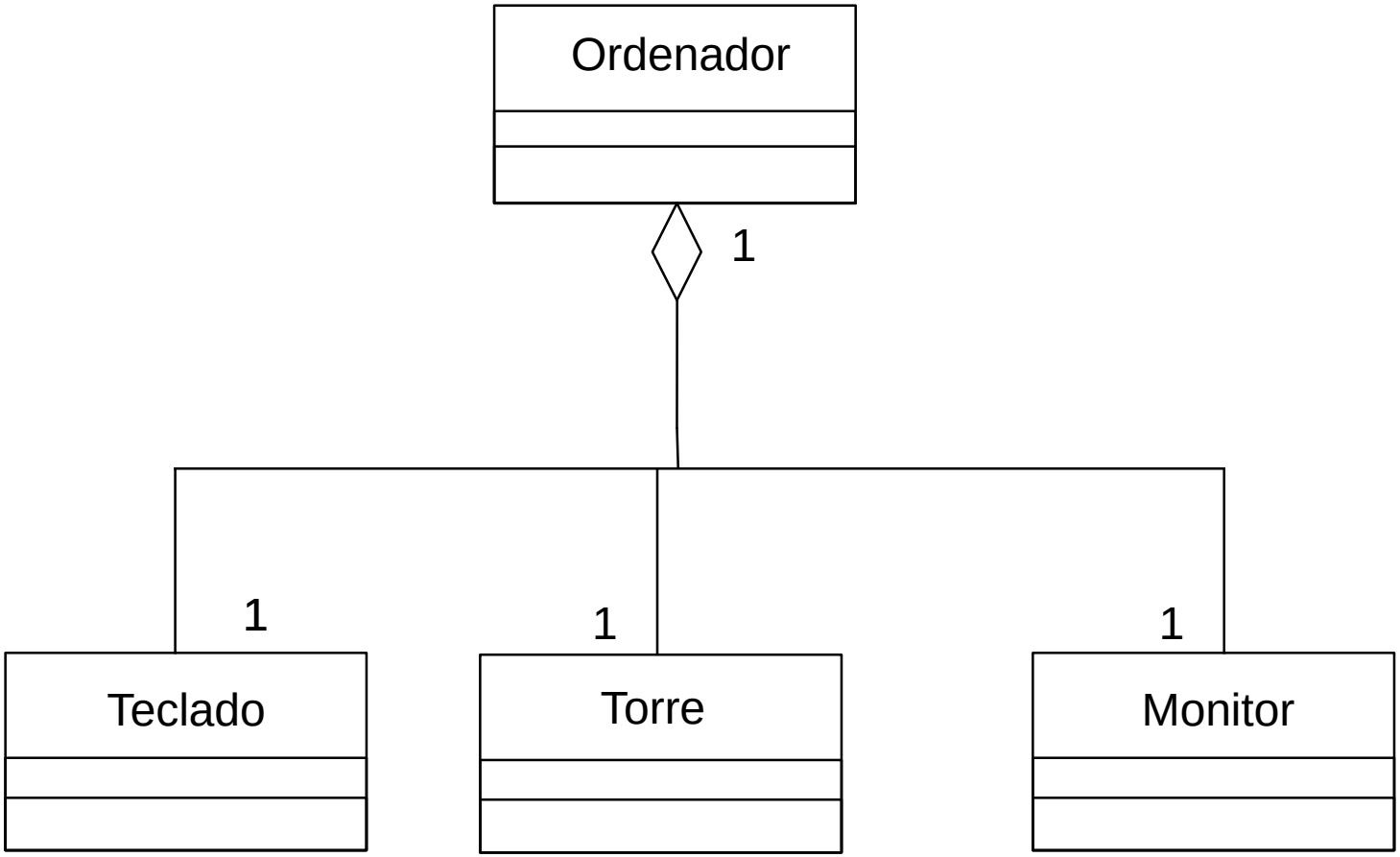
Conectores: Asociación reflexiva



Conectores: Agregación

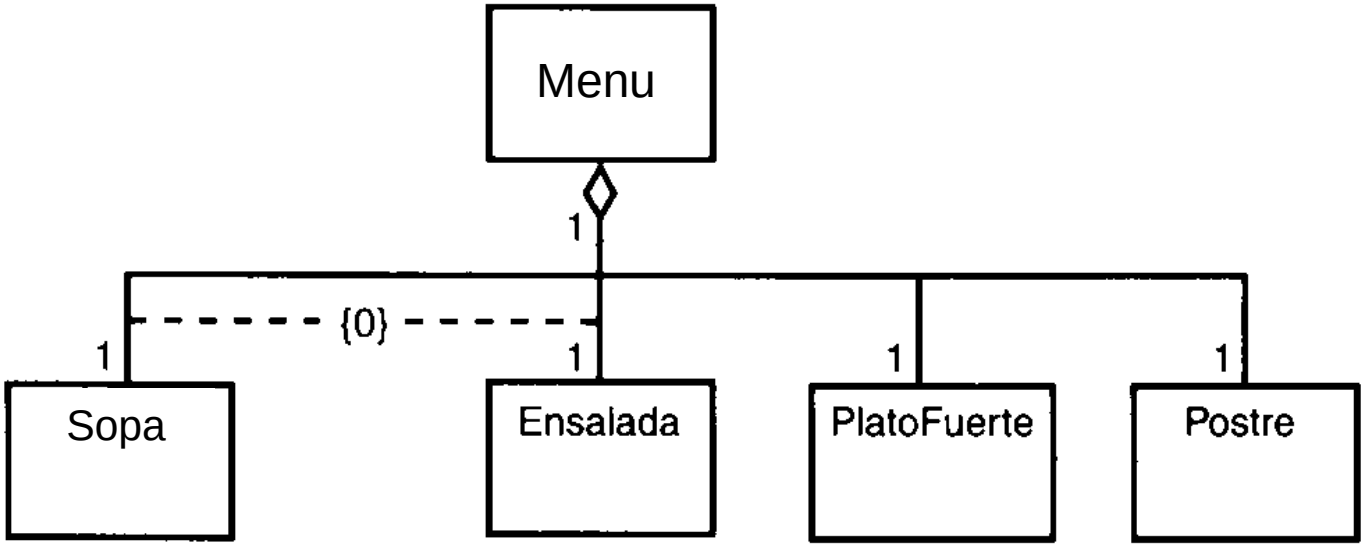
- En ocasiones una clase consta de otras clases
  - Los componentes y la clase que constituyen son una asociación que conforma un todo.
  - Si el objeto base desaparece no desaparecen los objetos incluidos
- Representa una relación parte_de entre objetos.
  - Se utiliza un rombo sin relleno en la parte todo (objeto compuesto) y se une con una línea a los componentes.

Conectores: Agregación



Conectores: Agregación

- A veces, el conjunto de componentes de una agregación se establece dentro de una relación O

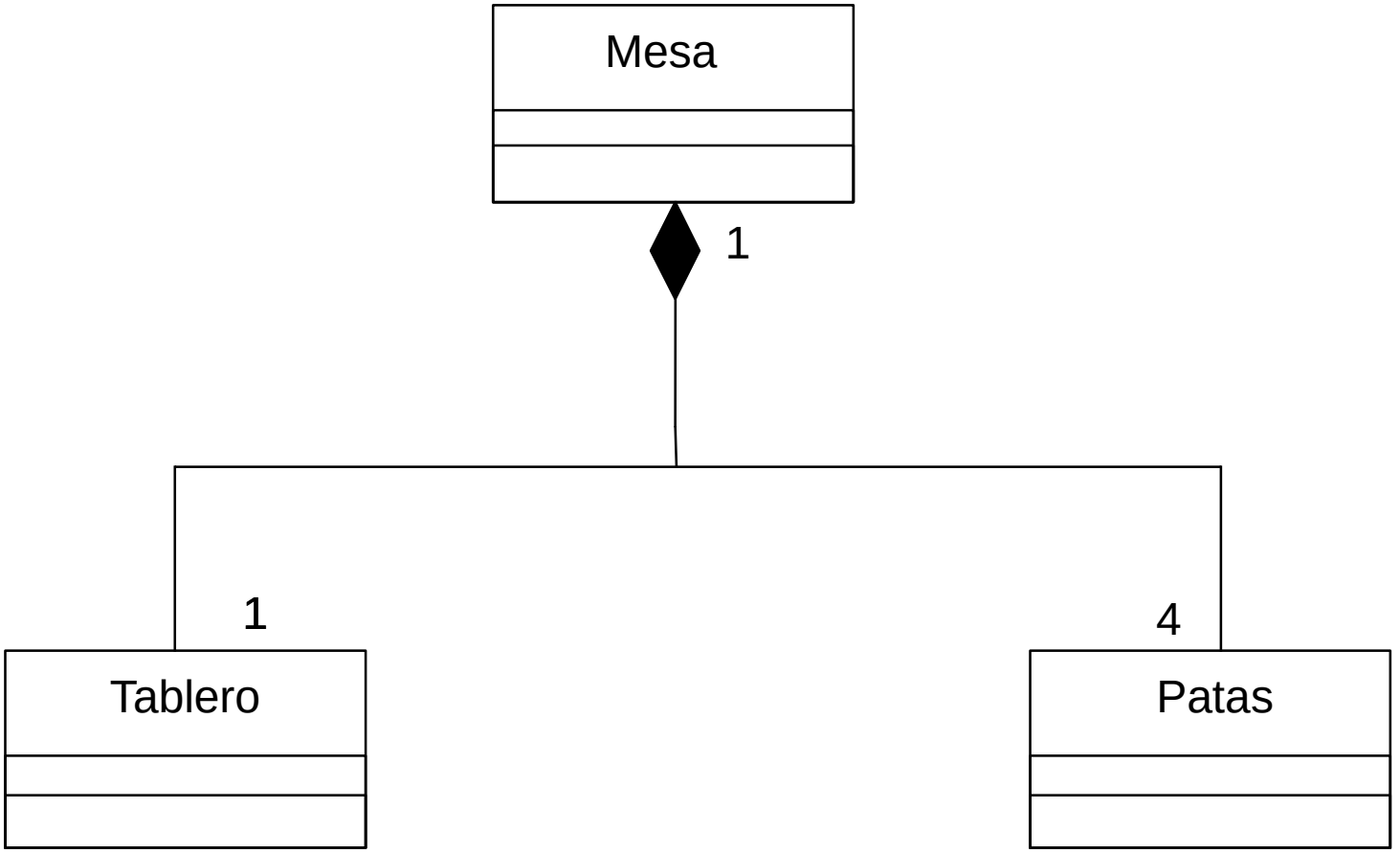


Conectores: Composición

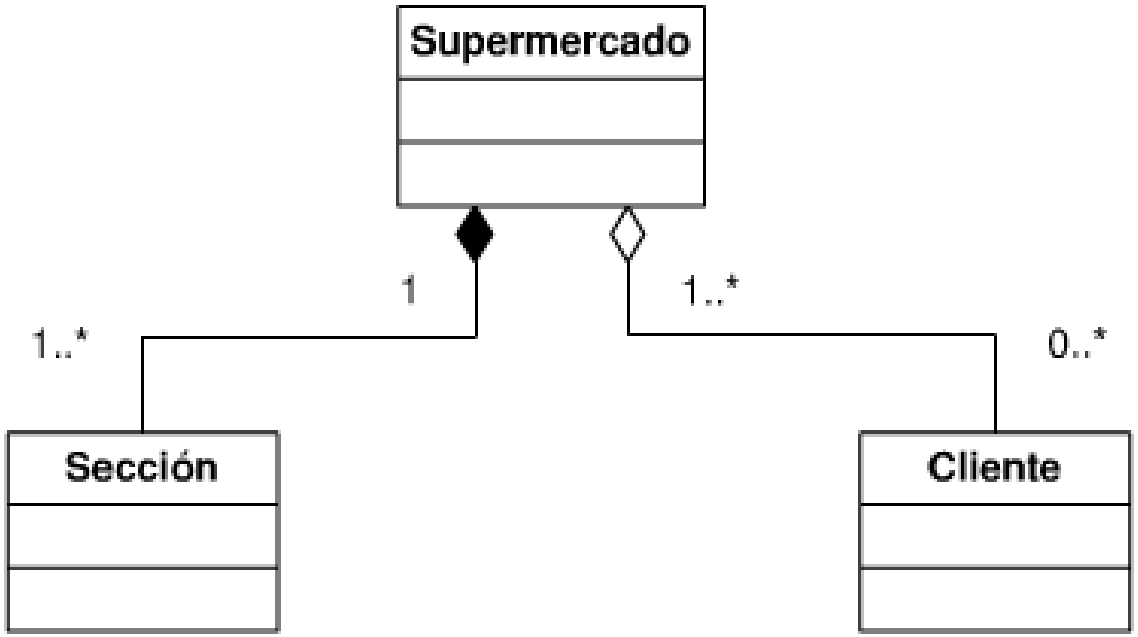
- Es un tipo representativo de una agregación
- Cada componente dentro de una composición puede pertenecer solamente a un todo
- El tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye
  - El objeto incluido sólo existe mientras exista el objeto base
  - El objeto base se construye a partir de los objetos incluidos pero no podría existir sin ellos

Conectores: Composición

- El símbolo de una composición es el mismo que una agregación, excepto que el rombo está relleno.



Conectores: Agregación vs Composición



Ejercicio

- Un libro de direcciones de email está conformado de múltiples contactos y grupos de contacto; un contacto se puede incluir en más de un grupo de contacto.
- Si se elimina un libro de direcciones, todos los contactos y grupos de contactos se eliminarán también;
- Si se elimina un grupo de contacto, ningún contacto se eliminará.

Ejercicio

- Indicar si las siguientes relaciones son de agregación o composición y cuál es la cardinalidad de la relación:
  - Caso1. Un barco puede pertenecer a 0 o 1 flotas y una flota debe tener al menos un barco
  - Caso2. Relación entre la clase libro y la clase capítulo de libro
  - Caso3. Relación entre la clase matrimonio y la clase persona
  - Caso4. Un sitio web está formado por varias páginas web

Agregación vs composición

	Agregación	Composición
Representación	Rombo blanco	Rombo negro
Varias asociaciones comparten los componentes	Sí	No
Destrucción de los componentes al destruir el compuesto	No	Sí
Cardinalidad a nivel de compuesto	Cualquiera	0..1, o 1

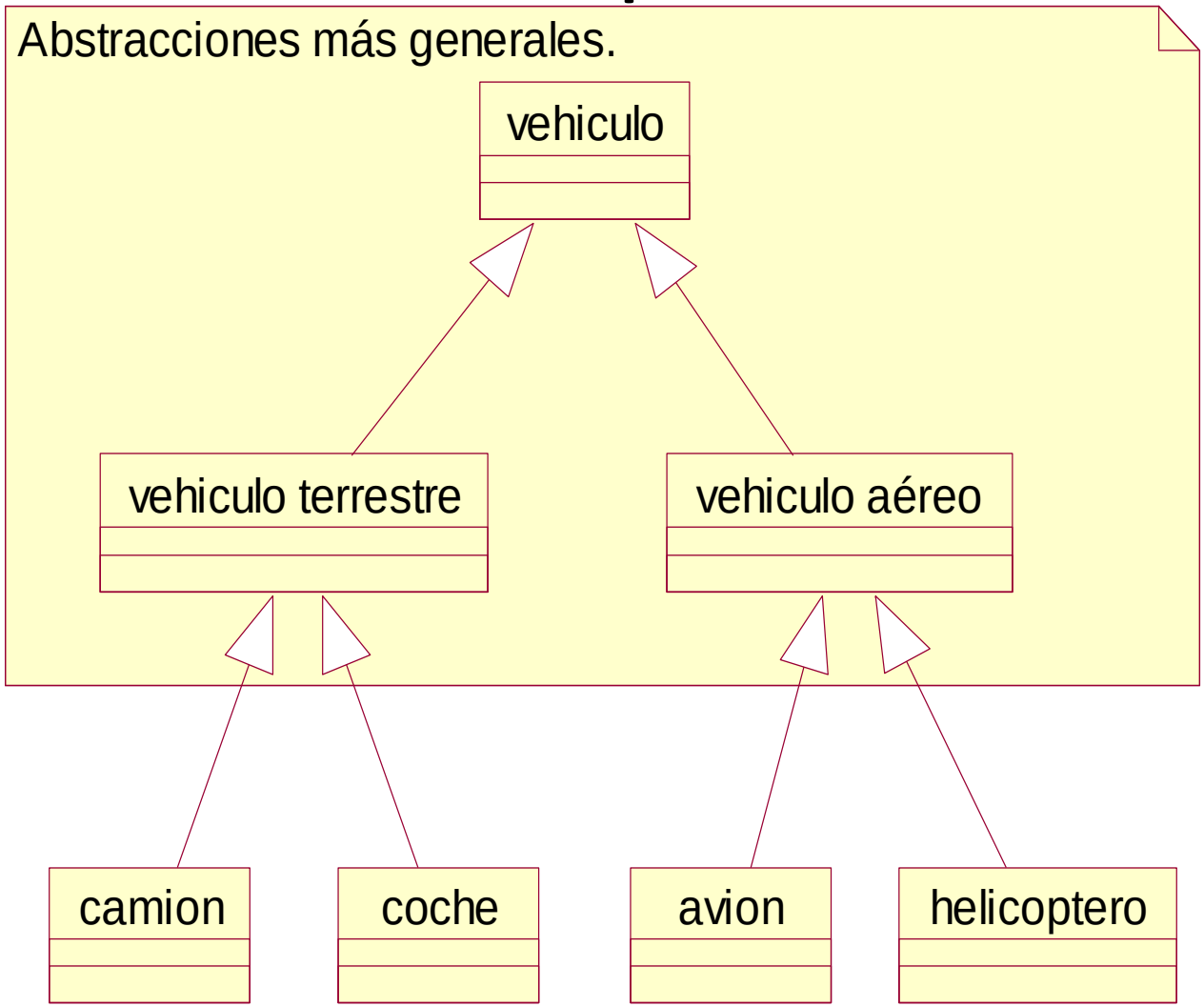
Conectores: Generalización/Especialización

- Permiten gestionar la complejidad mediante un ordenamiento taxonómico.
- Se obtiene usando los mecanismos de abstracción de Generalización y/o Especialización.
- La Generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general

Conectores: Generalización/Especialización

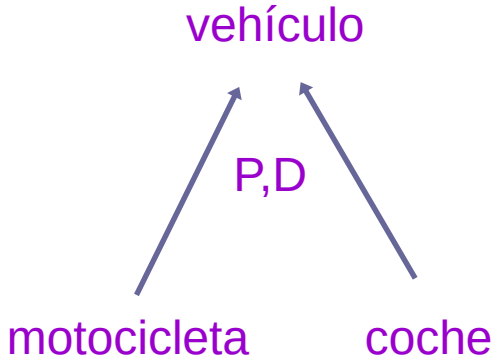
- Nombres usados:
  - clase padre - clase hija, superclase - subclase, clase base - clase derivada
- Las subclases heredan características de sus superclases
  - Atributos y operaciones (y asociaciones) de la superclase están disponibles en sus subclases.

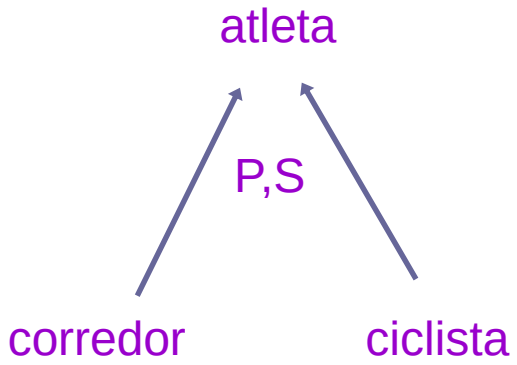
Conectores: Generalización/Especialización



Conectores: Generalización/Especialización

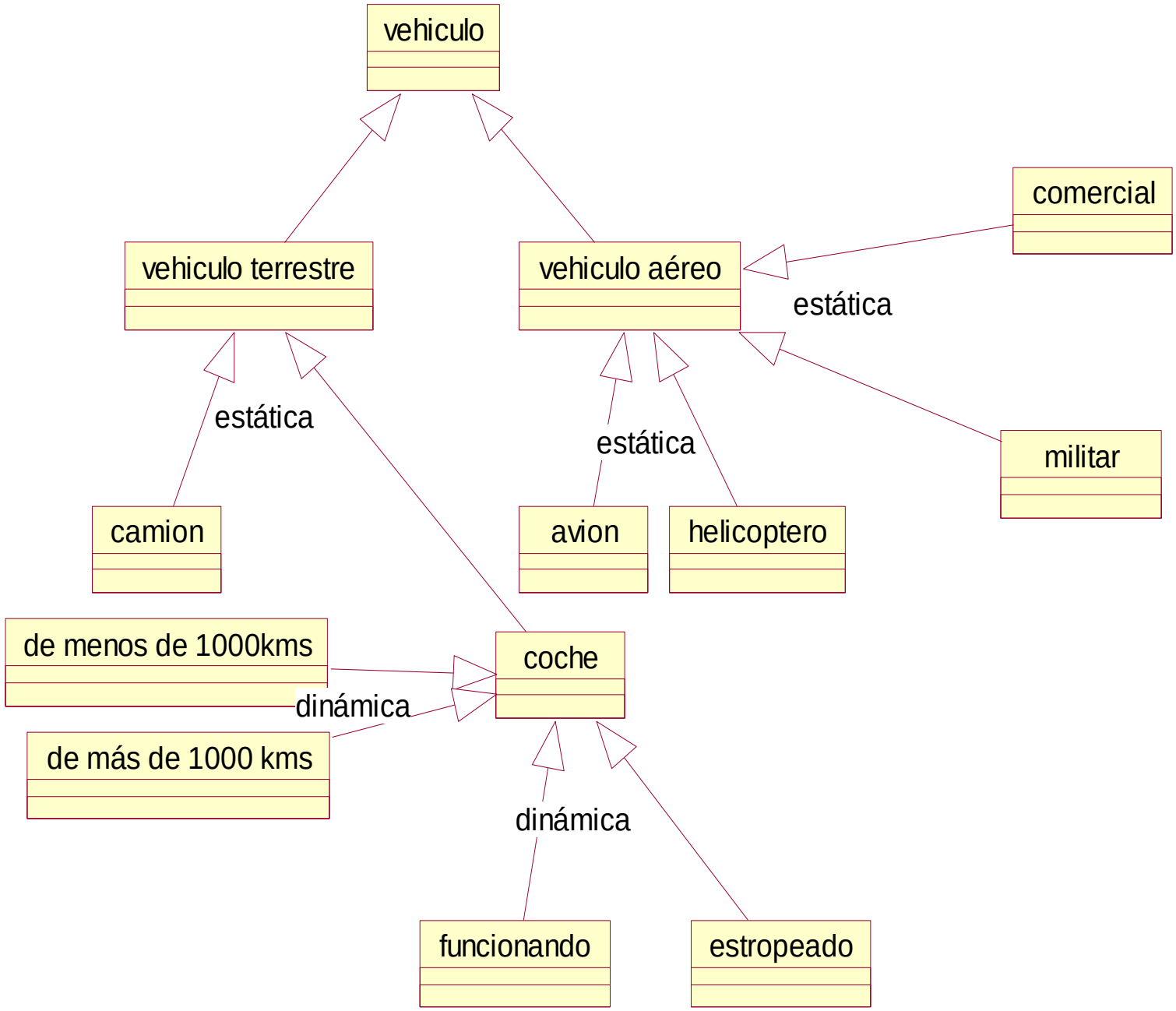
- Caracterización generalización
  - Cómo se clasifican los objetos
  - total / parcial
    - Todos / no todos los objetos pertenecen a una clase especializada
    - ¿Todos los vehículos son o coches o motos?
  - disjunta / solapada
    - Los conjuntos especializados son disjuntos / no disjuntos
    - Un vehículo, ¿puede ser coche y moto a la vez?





Conectores: Generalización/Especialización

- Se puede particionar el espacio de objetos:
  - Clasificación estática
- Se puede particionar el espacio de estados de los objetos:
  - Clasificación dinámica
- En ambos casos se recomienda considerar generalizaciones disjuntas

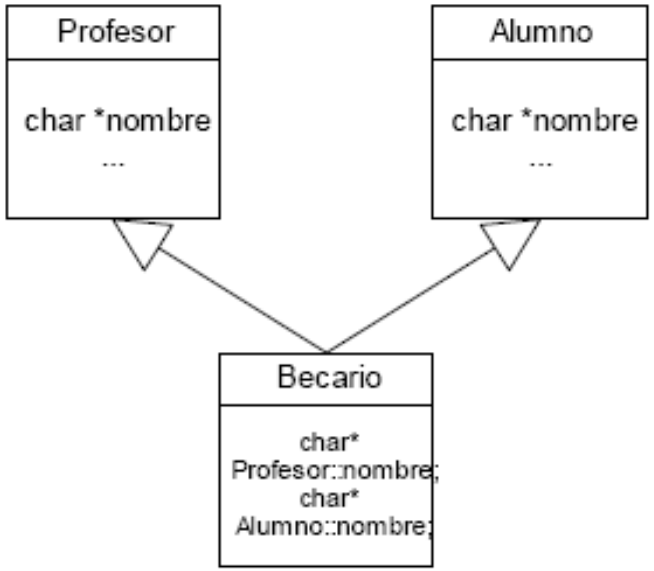

vehiculo
vehiculo terrestre
vehiculo aéreo
comercial
estática
estática
estática
militar
camion
avion
helicoptero
de menos de 1000kms
coche
dinámica
de más de 1000 kms
dinámica
funcionando
estropeado

Conectores:
Herencia múltiple

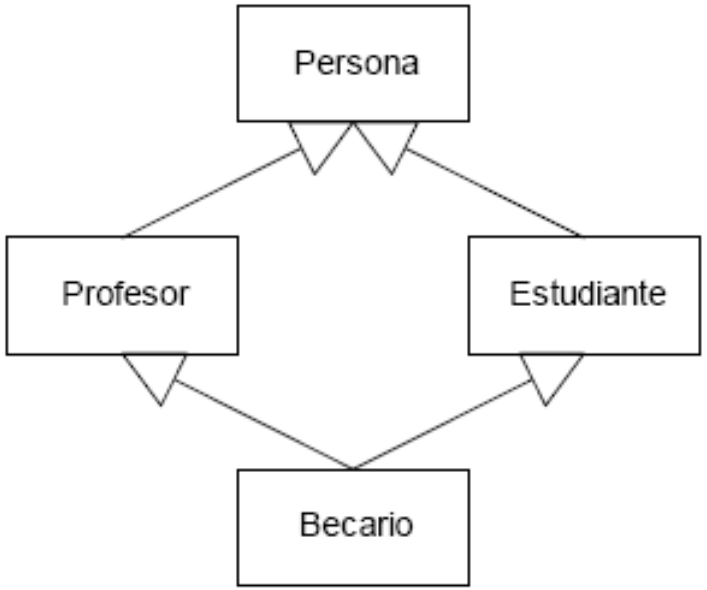
- Se presenta cuando una subclase tiene más de una superclase.
  - Debe manejarse con precaución.
    - Algunos problemas son el conflicto de nombre y el conflicto de precedencia.
  - Se recomienda un uso restringido y disciplinado de la herencia.
    - Java, C# y Ada 95 simplemente no ofrecen herencia múltiple

Conectores: Herencia múltiple

Colisión de nombres de diferentes superclases

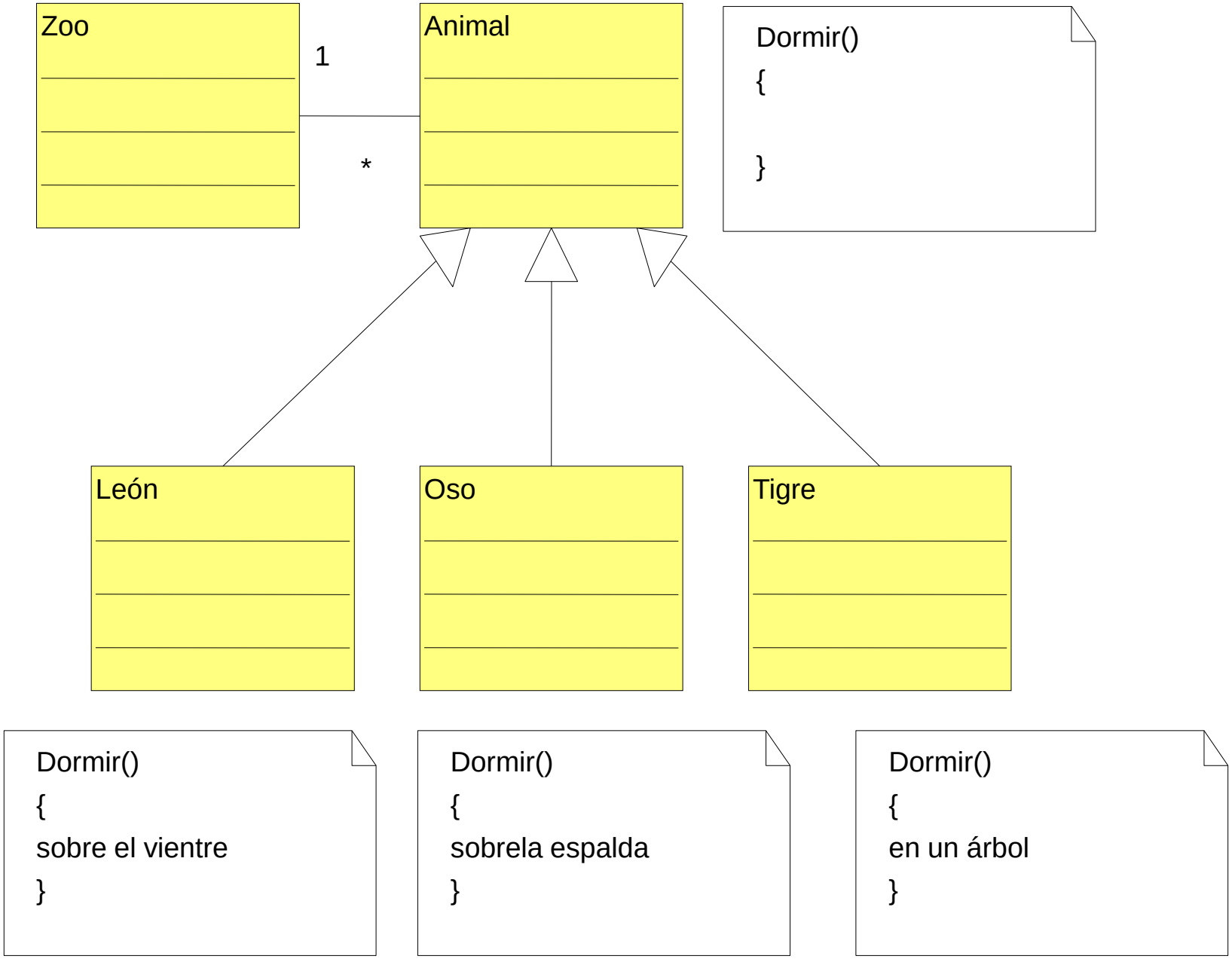


Herencia repetida

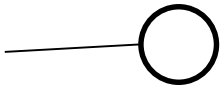


Conectores:
Herencia y Polimorfismo

- Se refiere a que una característica de una clase puede tomar varias formas.
- Representa la posibilidad de desencadenar operaciones distintas en respuesta a un mismo mensaje.
- Cada subclase hereda las operaciones pero tiene la posibilidad de modificar localmente el comportamiento de estas operaciones.


Zoo
1
*
Animal
Dormir()
{
}
León
Oso
Tigre
Dormir()
{
sobre el vientre
}
Dormir()
{
sobrela espalda
}
Dormir()
{
en un árbol
}

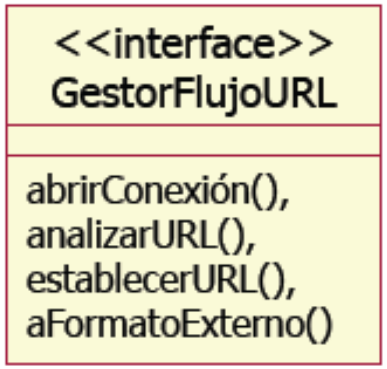
Conectores: Interfaces



- Una interfaz proporciona un conjunto de operaciones (métodos) que especifican cierto aspecto de la funcionalidad de una clase
- Permite que clases que no están estrechamente relacionadas entre sí deban tener el mismo comportamiento
- La implementación de una interfaz es un contrato que obliga a la clase a implementar todos los métodos definidos en la interfaz

Conectores: Interfaces

- Una vez creadas varias clases puede darse el caso de que no pertenezcan a una Clase Principal, pero en su comportamiento deben incluir algunas de las mismas operaciones
- Se declaran mediante la clase estereotipada <<interface>>



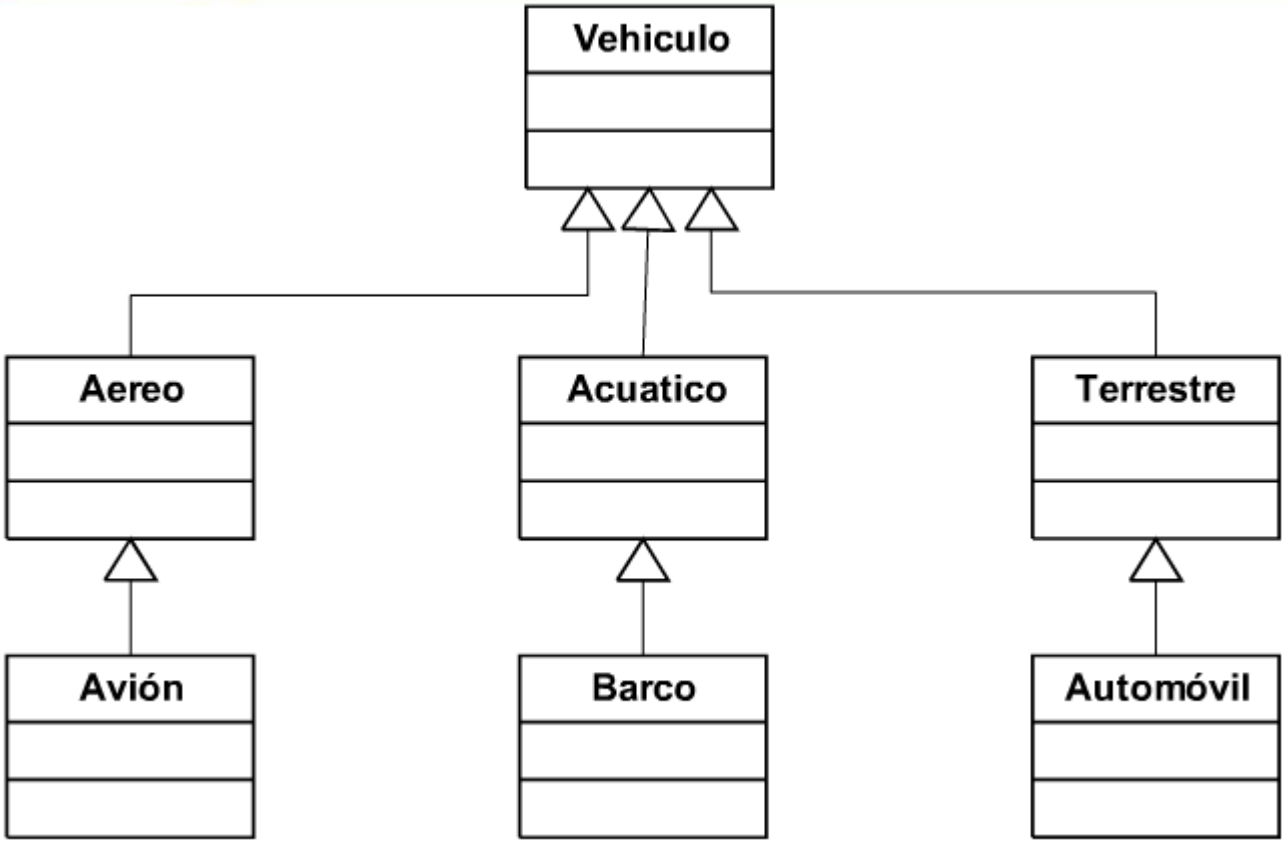
Conectores: Interfaces

- Diferencia entre herencia e interfaz
- Una interfaz es una clase *abstracta*
  - No tiene código ejecutable
- Cuando se usa la herencia se da a entender que una cosa se puede concebir como un tipo de otra: “es un”
  - Por ejemplo, una moto **“es un”** tipo de vehículo

Interfaces

- Un control remoto envía señales infrarrojas a distintos dispositivos:
  - Mando de una TV (subir o bajar volumen…)
  - Mando de la puerta de un garaje (subir o bajar la puerta)
- Los dispositivos anteriores no están relacionados pero comparten la característica de subir o bajar
- Un mando de la puerta de un garaje no es un tipo de mando de TV ni viceversa
- Mediante una interfaz se puede modelar las partes similares de dos clases que no tienen relación entre sí

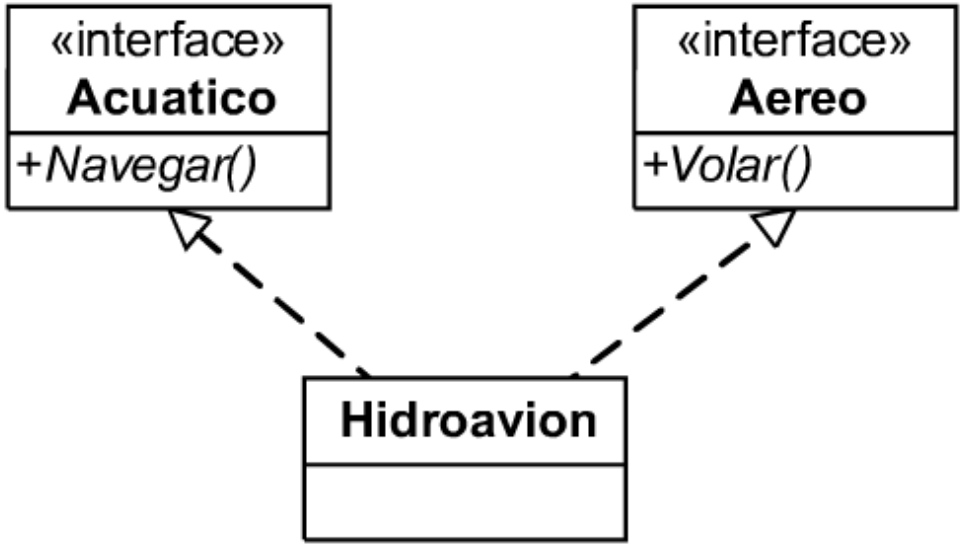
Conectores: Interfaces



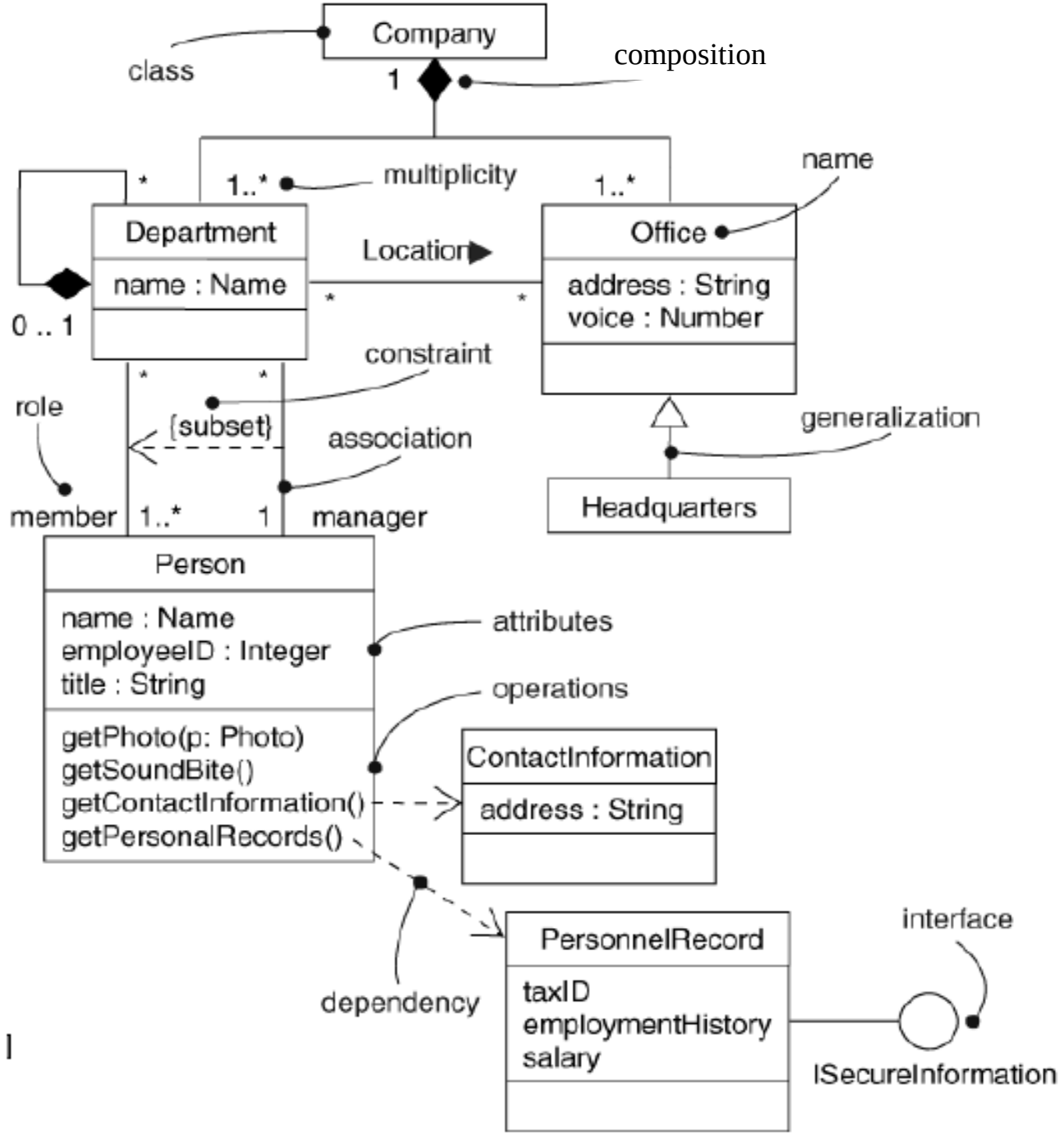
¿De qué clase heredaría hidroavión?

Conectores: Interfaces

- Se crean las interfaces que definen el comportamiento
- Hidroavión debería definir los comportamientos de cada una de las interfaces que implemente



- Un ejemplo de este tipo de diagrama puede ser visto en la figura de la siguiente transparencia.
- En este diagrama podemos observar cómo una determinada compañía (clase) tiene (relación de composición) oficinas y departamentos. Cada departamento está relacionado (relación de asociación) con una o más personas (sus miembros), de las cuales una es además jefe del departamento. Por otro lado, las oficinas pueden ser (relación de herencia) oficinas centrales. Cada clase tiene atributos y métodos. Algunos métodos requieren para su ejecución acceder a información acerca de objetos de otras clases con las que no tiene una relación directa (en este caso el método get-PersonalRecords() necesita información acerca de determinados objetos de la clase PersonnelRecord). Esta necesidad se plasma en el diagrama de clases mediante relaciones de dependencia.


Company
class
composition
1
multiplicity
1..*
1..*
name
*
Department
name : Name
Location
Office
address : String
voice : Number
0 .. 1
*
*
constraint
role
{subset}
association
generalization
member
1..*
1
manager
Headquarters
Person
name : Name
employeeID : Integer
title : String
attributes
operations
getPhoto(p: Photo)
getSoundBite()
getContactInformation()
getPersonalRecords()
ContactInformation
address : String
dependency
PersonnelRecord
taxID
employmentHistory
salary
interface
ISecureInformation

Ejercicio

- Representa mediante un diagrama de clases la siguiente especificación:
  - Una compañía necesita almacenar información sobre sus empresas, sus empleados y sus clientes (ambos se caracterizan por su nombre y edad).
  - Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.
  - De los clientes además se necesita conocer su teléfono de contacto.

Ejercicio

- Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (novela, teatro, poesía, ensayo), editorial, año y autor.
- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, con retraso o en reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, por cada día de retraso, se impone una “multa” de dos días sin posibilidad de coger un nuevo libro.

Bibliografía

- UML gota a gota. Martin Fowler
- Ingeniería del software. Ian Sommerville
- UML distilled. Martin Fowler