1ºDAM-S 2017/2018 E.D.

DIAGRAMA DE CLASES

Vicente Monfort

Diagramas de Clases

Un diagrama de clases es un tipo de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos) y las relaciones entre los objetos.

Los diagramas de clase son un tipo de diagrama estructural.

Los diagramas estructurales describen componentes del sistema y sus relaciones.

Los diagramas de clase muestran las clases que componen el sistema, su estructura interna y sus relaciones con otras clases.

El diagrama de clase es una representación estática del sistema.

Clases y objetos

Una clase es una descripción de un conjunto de objetos con las mismas propiedades (atributos) y el mismo comportamiento (operaciones).

Tanto con la programación orientada a objetos (POO) como en UML, una clase es una plantilla que representa un concepto del mundo real y se utiliza para crear objetos. Por ejemplo, podemos abstraer todas las lavadores en una clase Lavadora junto con sus atributos (marca, modelo, consumo, velocidad) y sus operaciones (lavar, enjuagar, centrifugar).

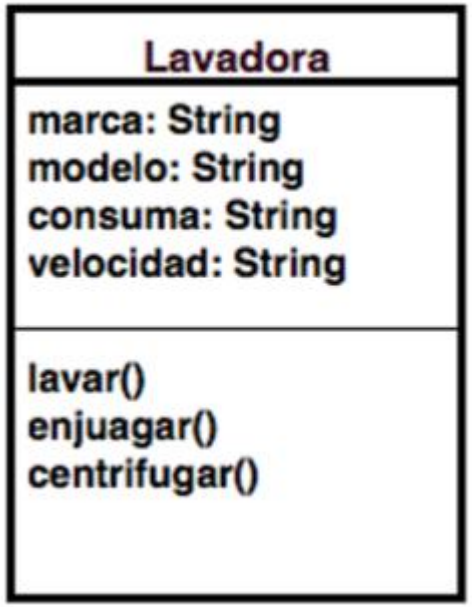
Clases y objetos

A partir de una clase se pueden crear objetos, también llamados instancias de la clase.

Los objetos son concreciones de una clase: todos los objetos de una clase comparten las mismas operaciones, pero sus atributos pueden tener valores distintos.

Los atributos de una clase suelen llamarse variables miembro.

Las operaciones de UML equivalen a métodos o funciones miembro en cualquier lenguaje de programación.

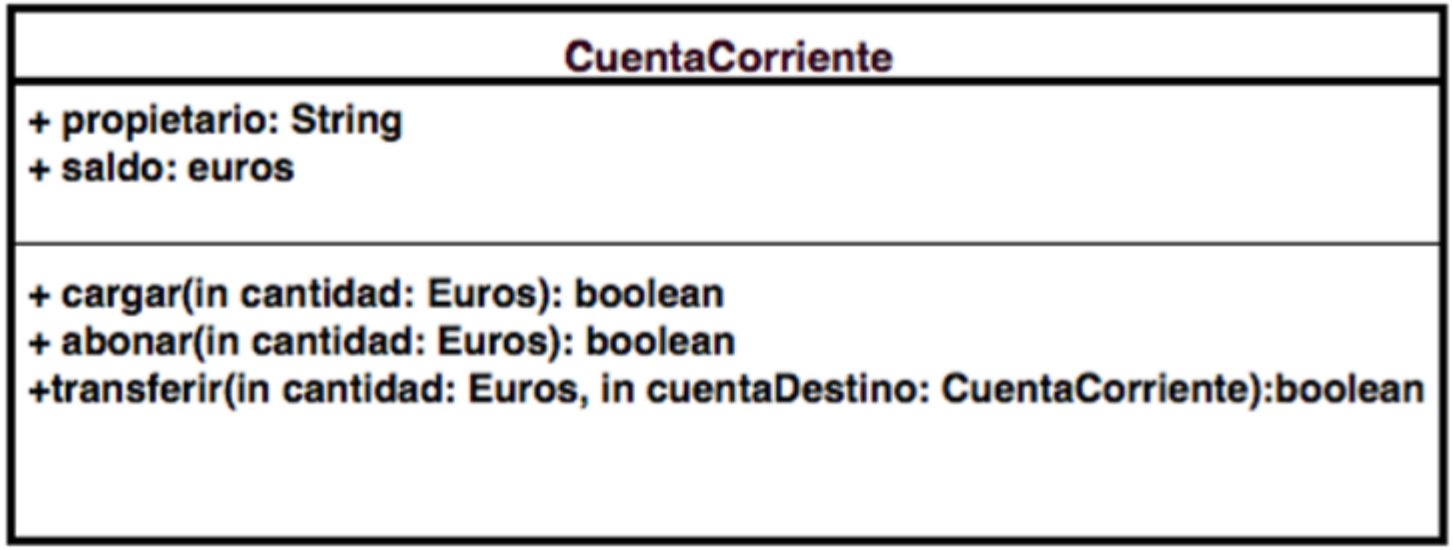


Ejemplo CuentaCorriente

Con respecto a los atributos tenemos el nombre del atributo seguido de su tipo (aunque el tipo está asociado al lenguaje de programación que se utilizará).

Con respecto a la lista de operaciones podemos añadir la dirección del parámetro que puede ser: **in** (entrada), **out** (salida), **inout** (bidireccional: entrada/salida).

En el ejemplo de CuentaCorriente se tienen 3 operaciones: cargar, abonar y transferir. La operación cargar recibe un único parámetro que será la cantidad de euros. La operación transferir toma 3 parámetros, ambos de entrada, y serán la cantidad de euros y la cuenta destino (que será de tipo CuentaCorriente).



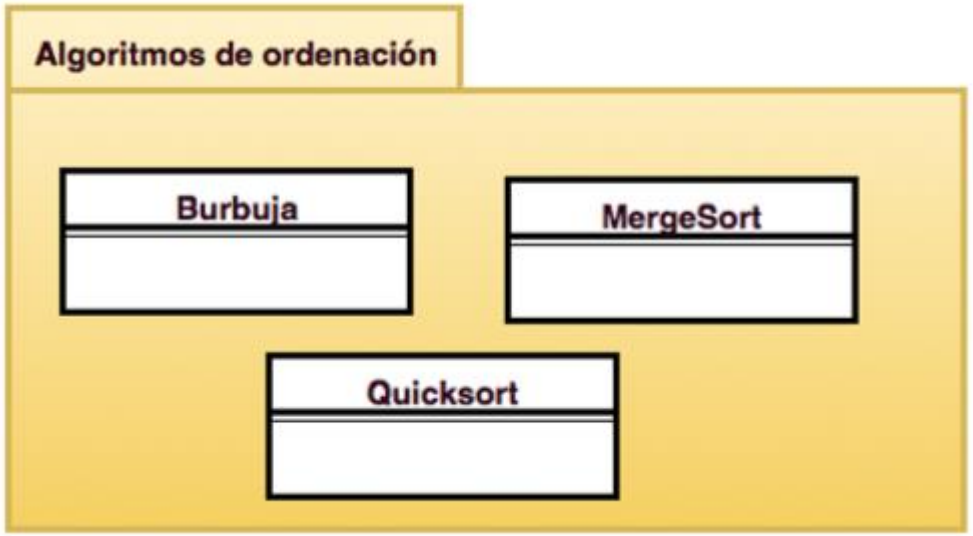
Paquetes

Si modelamos un sistema complejo, inevitablemente aparecerán muchas clases en el modelo.

Los paquetes (packages) son contenedores que permiten agrupar clases con características o funcionalidades comunes.

Un módulo o paquete es una construcción lógica para agrupar clases, asociaciones y generalizaciones.

En UML, un paquete se representa con un recuadro con una solapa donde se escribe el nombre del paquete y en el interior del recuadro se dibujan las clases pertenecientes al paquete.



Visibilidad

El acceso a los elementos (operaciones o atributos) desde otras clases depende de la visibilidad.

Cada elemento puede tener una visibilidad de entre estas cuatro posibles:

- Pública (**public**): un elemento es visible desde cualquier otra clase.
- Protegida (**protected**): un elemento protegido es visible sólo por elementos de su propia clase o las clases hijas.
- Privada (**private**):un elemento privado sólo es visible por elementos de su propia clase.
- Paquete (**package**): un elemento de paquete sólo es visible por otras clases que pertenezcan al mismo paquete.

Visibilidad

Sea cual sea la visibilidad, un objeto siempre podrá acceder a los elementos de su propia clase.

La visibilidad se representa en el diagrama mediante un símbolo antepuesto al nombre de la operación o atributo:

- +: atributo u operación pública
- #: atributo u operación protegida
- -: atributo u operación privada
- ~: paquete

Ejemplo Clase Alumno

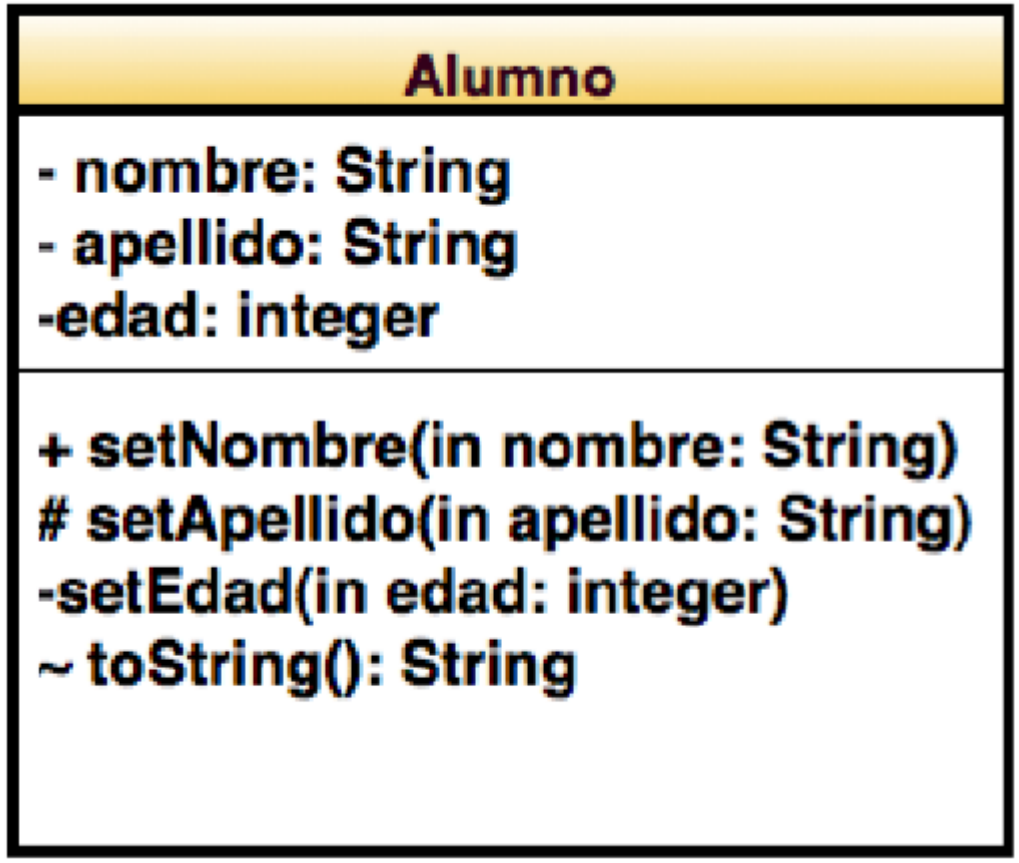
Atributos: nombre, apellidos y edad son privados.

El método setNombre es público

El método setApellido es protegido.

El método setEdades privado.

El método toString es aplicable a nivel de paquete.



Estereotipos

Los estereotipos son un mecanismo que permite extender el lenguaje UML añadiendo información extra a los elementos del diagrama.

Por ejemplo, el estereotipo <<Create>> aplicado a una operación indicaría que esa operación es el constructor de una clase.

Los estereotipos se pueden aplicar a cualquier elemento del diagrama (clases, operaciones, atributos, relaciones…) y se expresan entre comillas francesas colocando encima o cerca del nombre del elemento: <<estereotipo>>.

UML tiene algunos estereotipos ya predefinidos pero podemos inventarnos nuevos.

Relaciones entre clases

En una aplicación mínimamente compleja habrá varias clases cuyos métodos se llamarán unos a otros.

Para que un método de la clase A llame a otro método de la clase B, la clase A debe poseer una referencia a un objeto de la clase B.

Esta relación de posesión se representa con líneas que unen las distintas clases en el diagrama.

Las relaciones pueden ser:

- Asociación
- Agregación
- Composición
- Dependencia

Asociación

Es el tipo de relación **más frecuente** entre las clases.

Existe una relación de asociación entre ClaseA y ClaseB cuando en ClaseA hay un atributo de tipo ClaseB o viceversa.

Se representa una línea continua, que puede estar o no acabada en una flecha en “V”, según la navegabilidad.

Además se suelen indicar los roles y la multiplicidad.

Con respecto a la navegabilidad, si la flecha de la claseA apunta a la claseB, se lee que la claseA tiene una claseB y se dice que la asociación o navegabilidad es unidireccional.

Si no dibujamos ninguna flecha, se dice que la claseA tiene una claseB y la claseB tiene una claseA y se dice que la asociación o navegabilidad es bidireccional.

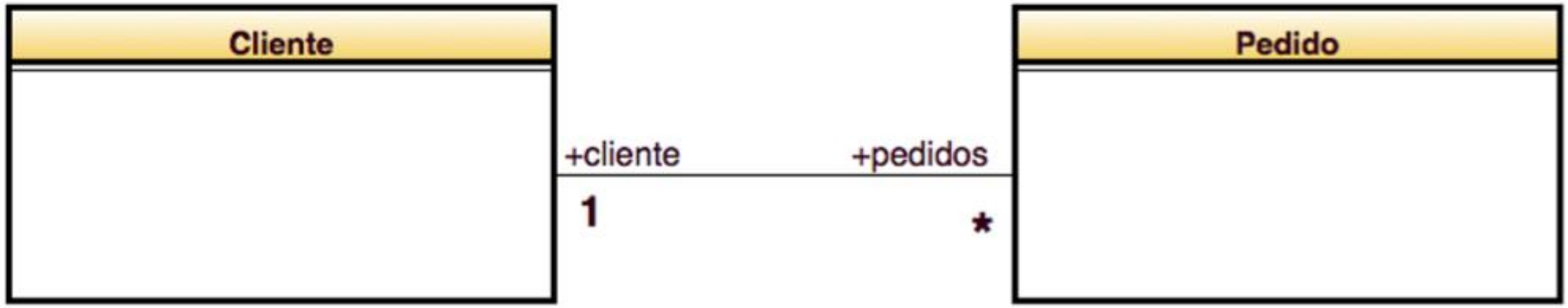
Asociación

En el siguiente ejemplo, el cliente guarda información sobre los pedidos y pedido tiene información sobre el cliente que lo realizó. La navegabilidad es bidireccional.

En concreto vemos que cada objeto de tipo cliente tendrá un atributo pedidos que contendrá una lista de los pedidos realizados.

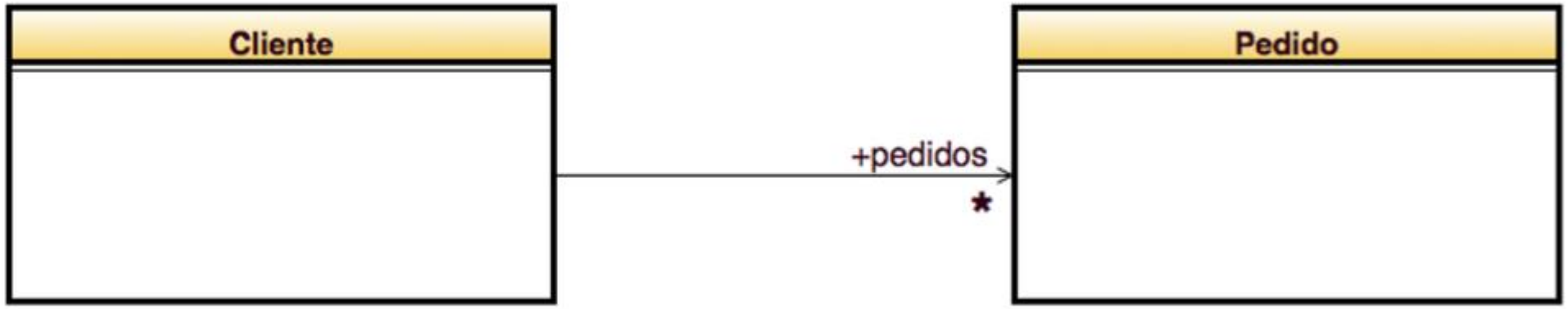
Por otra parte, los pedidos tendrán una referencia al cliente que lo realizó.

Si vemos el ejemplo, sabemos que un cliente puede tener un listado de pedidos debido a la multiplicidad representada con el asterisco(*).



Asociación unidireccional

Si queremos que nuestra aplicación guarde la lista de pedidos en el objeto Cliente y no a la inversa, usamos la navegabilidad unidireccional. En el siguiente ejemplo, los clientes almacenan información sobre los pedidos, pero los pedidos no tienen ninguna información sobre los clientes.



Asociación unidireccional

En este ejemplo, se modela la asociación a la inversa y los pedidos contienen información del cliente, pero los clientes no guardan pedidos.

Roles y multiplicidad

El nombre de los atributos se escribe en el lado contrario a la clase que los contiene.

Respecto a la multiplicidad se permite indicar cuántos objetos de una clase se pueden relacionar como mínimo y como máximo con objetos de otra clase.

Cuando las multiplicidades mínima y máxima son iguales, se suele representar con un único número.

La multiplicidad de tipo “muchos” se representa con un asterisco(*).

La multiplicidad de 0 a * se suele expresar simplemente como*

Cardinalidad de las asociaciones

La cardinalidad situada en un extremo de una asociación indica a cuántas instancias de la clase situada en ese mismo extremo está vinculada una instancia de la clase situada en el extremo opuesto.

Sino se especifica cardinalidad, esta valdrá 1.

Especificación	Cardinalidades
0..1	Cero o una vez
1	Únicamente una vez
*	De cero a varias veces
1..*	De una a varias veces
M..N	Entre M y N veces
N	N veces

Agregación y composición

La agregación y la composición son “asociaciones que representan una relación entre un todo y sus partes”. Se puede leer como “claseB es un componente de la claseA”, o también “ClaseA está compuesto por claseB”.

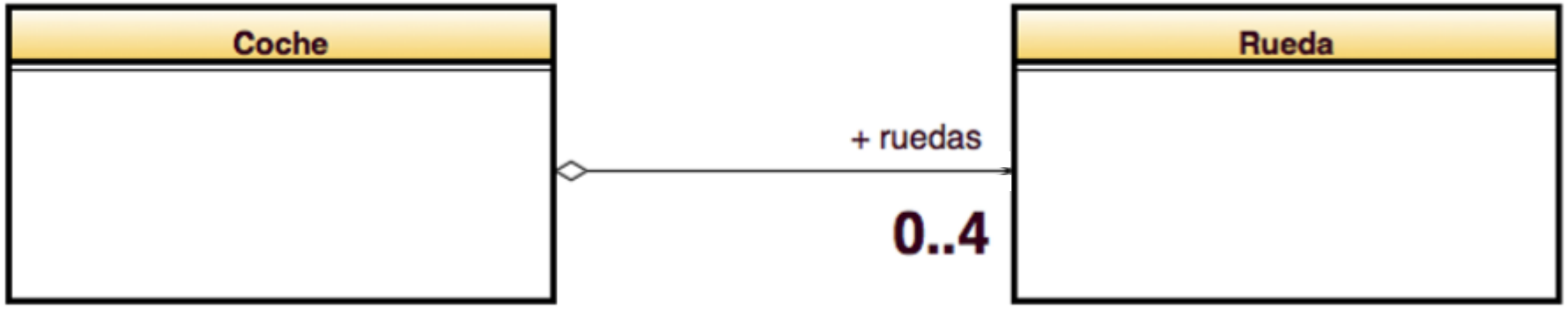
Las diferencias entre ellas son:

- En la **agregación**, los objeto “parte” pueden seguir existiendo independientemente del objeto “todo”.
- El la **composición**, por el contrario, la vida de los objetos compuestos están íntimamente ligada a la del objeto que los compone, de manera que si el “todo” se destruye, las “partes” también se destruyen.

Agregación

La agregación se representa uniendo dos clases (todo/parte) con una línea continua y poniendo un rombo hueco en la clase “todo”.

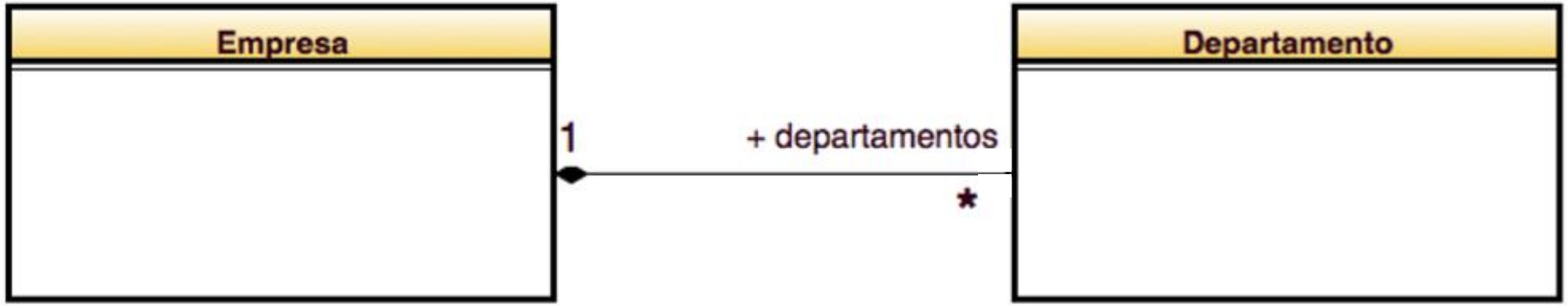
La composición se representa igual que la agregación pero con el rombo relleno.



En el ejemplo, hay una agregación donde un coche está compuesto por ruedas. Las ruedas pueden existir por sí mismas (ya que es una agregación).

Composición

En el siguiente ejemplo, vemos una composición (representado con el rombo relleno) donde se representa que una empresa está compuesta por departamentos. Los departamentos por sí mismos no pueden existir, ya que deben estar ligados a una empresa, por este motivo, la relación es de composición.

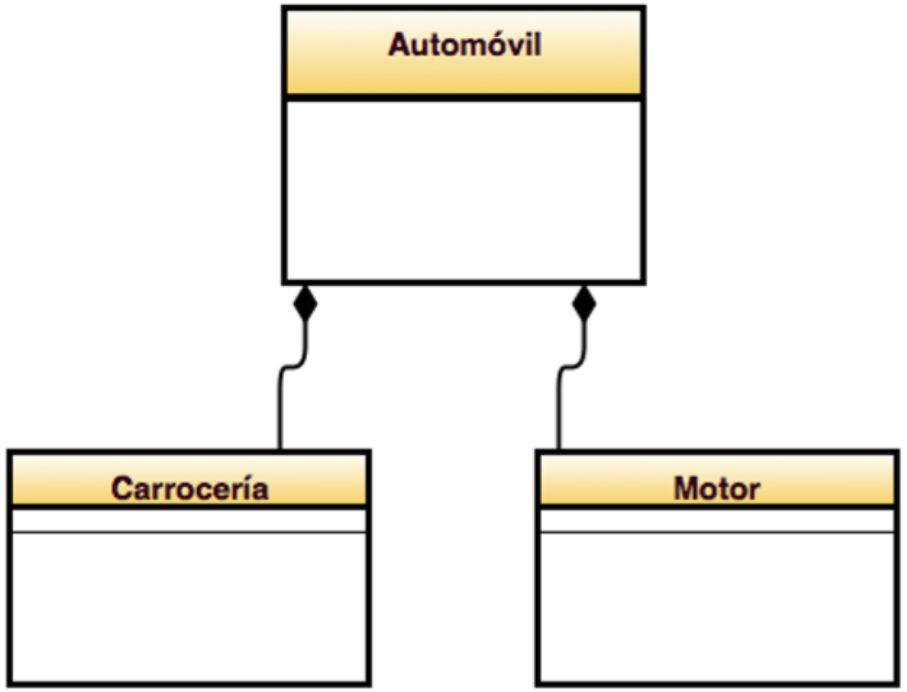


Composición

En el siguiente ejemplo, vemos que un caballo, entre otras cosas, está compuesto por un cerebro. El cerebro no se comparte. La muerte del caballo comporta la muerte del cerebro. Se trata, por tanto, de una asociación de composición.

Composición

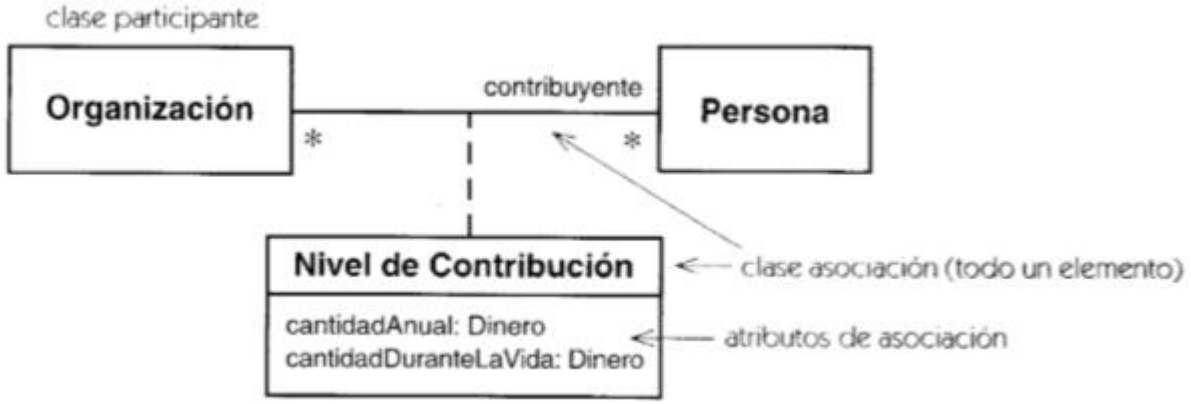
En la siguiente imagen, vemos un ejemplo de composición donde un automóvil está compuesto de carrocería y de un motor.



Clase asociación

Una clase puede tener atributos por sí misma, en cuyo caso es una asociación y una clase, o sea una clase asociación.

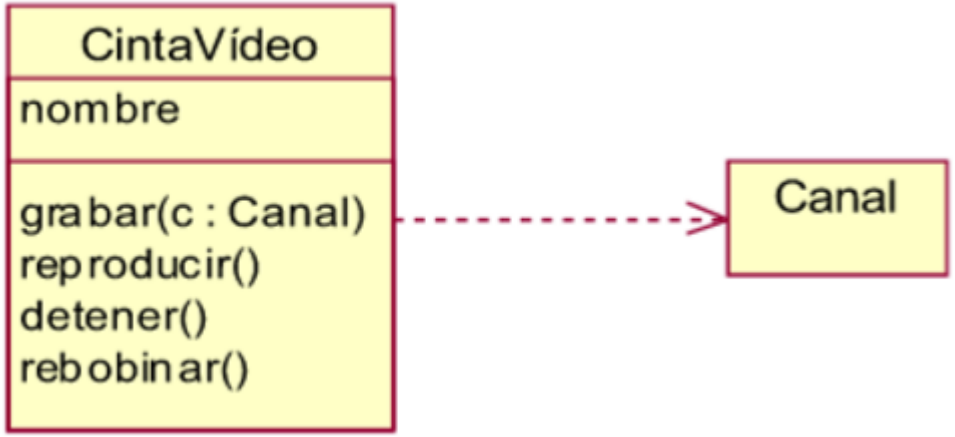
Se utiliza cuando queremos 'especificar más detalle en la asociación'



Dependencia

Una dependencia es una relación de uso en la que un cambio en uno de los términos puede afectar a otra clase.

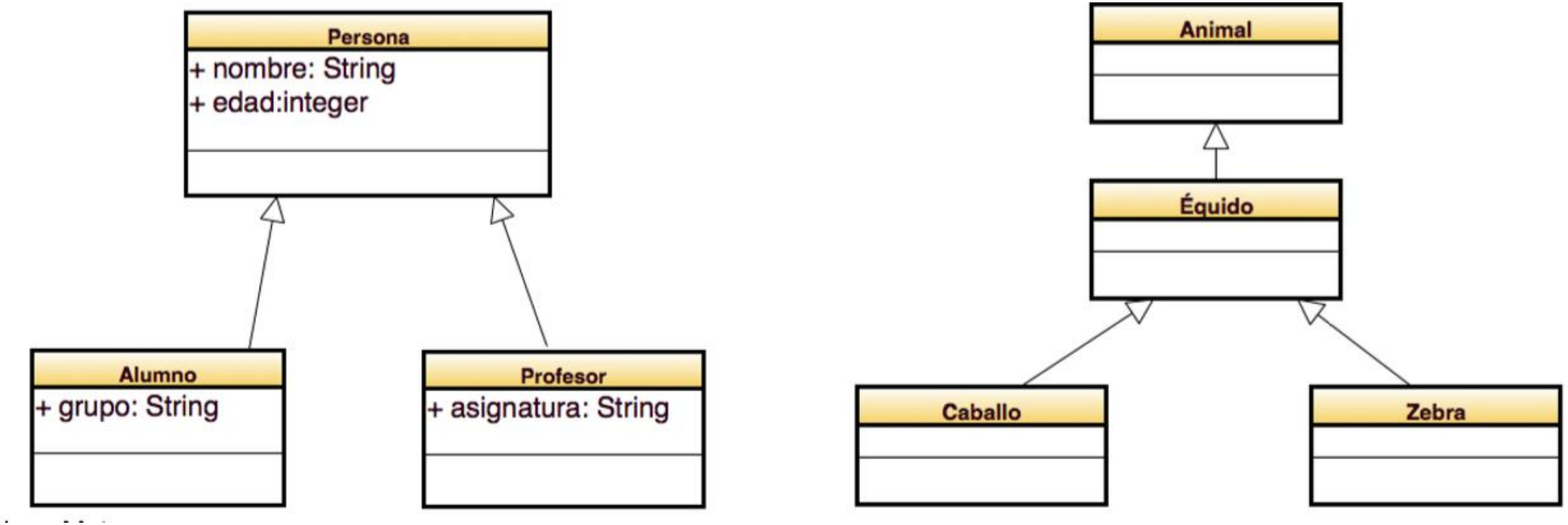
La relación de dependencia es aquella relación en la que **una clase necesita a otra** para poder funcionar. Un ejemplo de dependencia que va desde de la clase que tiene la operación hasta la clase usada como parámetro.



Generalización/Especialización

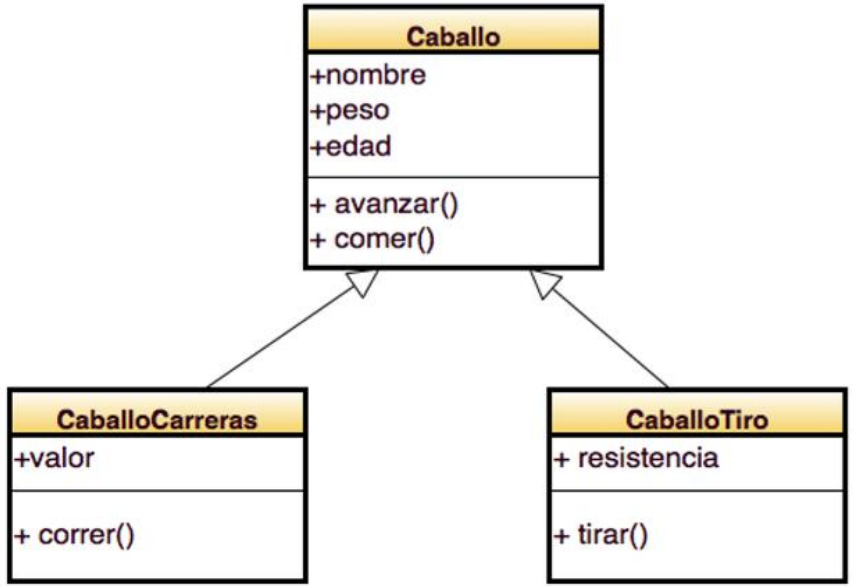
La relación de generalización entre dos clases indica que una de las clases (la subclase es una especialización de la otra (superclase). Si la claseA es la superclase y la claseB es la subclase, la generalización se podría leer como “claseB es una claseA”, o “claseB es un tipo de claseA”. En la mayoría de lenguajes de programación, la generalización se conoce como herencia. A la subclase se le puede llamar clase hija o clase derivada. A la superclase se le puede llamar clase padre o clase base.

Generalización/Especialización



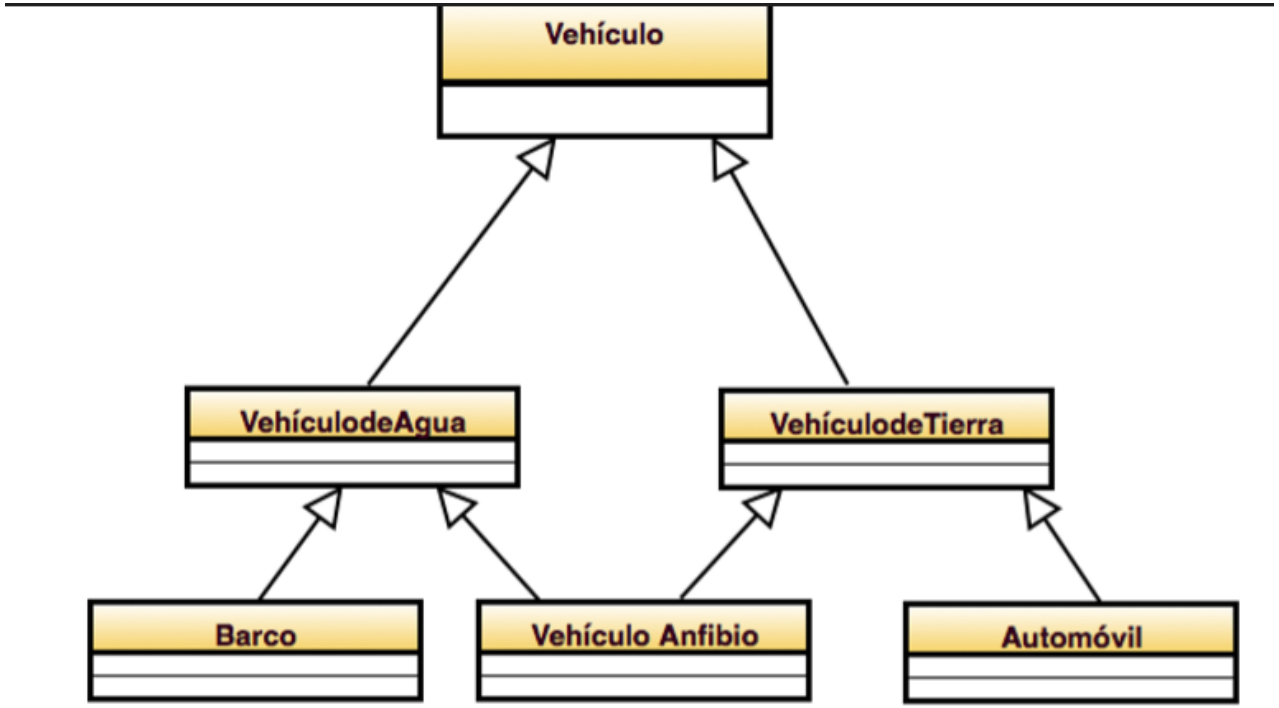
Herencia

Las instancias de una clase son también instancias de su superclase o superclases. Por consiguiente, aprovechan los atributos y métodos definidos en la superclase, además de los atributos y métodos introducidos en la clase. Una clase hereda los atributos y métodos de las superclases para que sus instancias se beneficien de ellos. Recordemos que los atributos y métodos privados de una superclase se heredan en sus subclases, pero no son visibles en ellas. En la herencia, los métodos pueden redefinirse en la subclase.



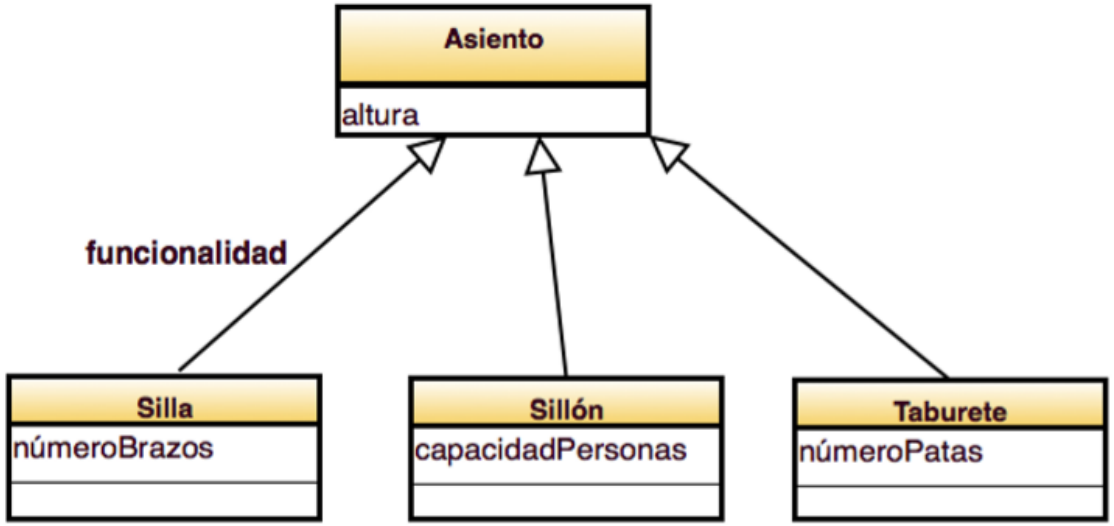
Herencia múltiple

La herencia múltiple permite a una clase tener más de una superclase y la herencia de todos sus ancestros.



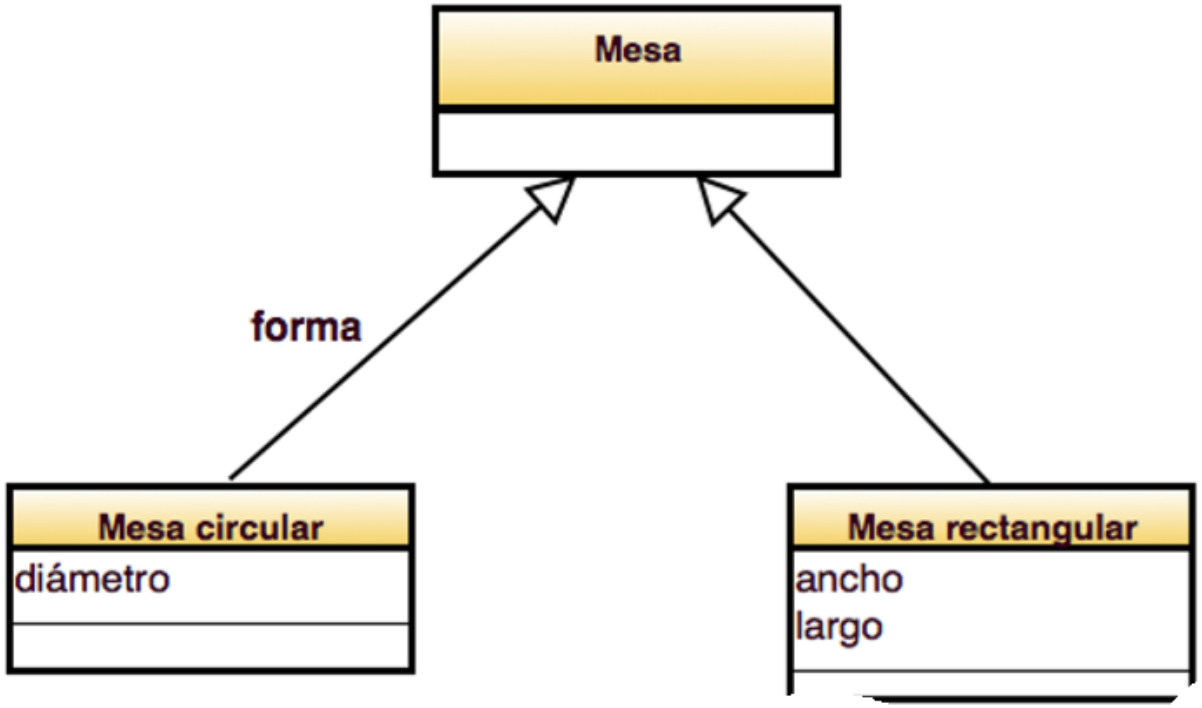
Discriminador

Un discriminador indica el atributo a la superclase cuyo valor distingue a las subclases. El discriminador indica cual propiedad común de las subclases se generaliza en la superclase (sólo una propiedad debe ser discriminadora a la vez). Ejemplo: Un asiento se puede discriminar según su funcionalidad, en silla, taburete o sillón.



Discriminador

Una mesa se puede discriminar según su forma, en mesa circular o mesa rectangular.



Clases concretas y abstractas

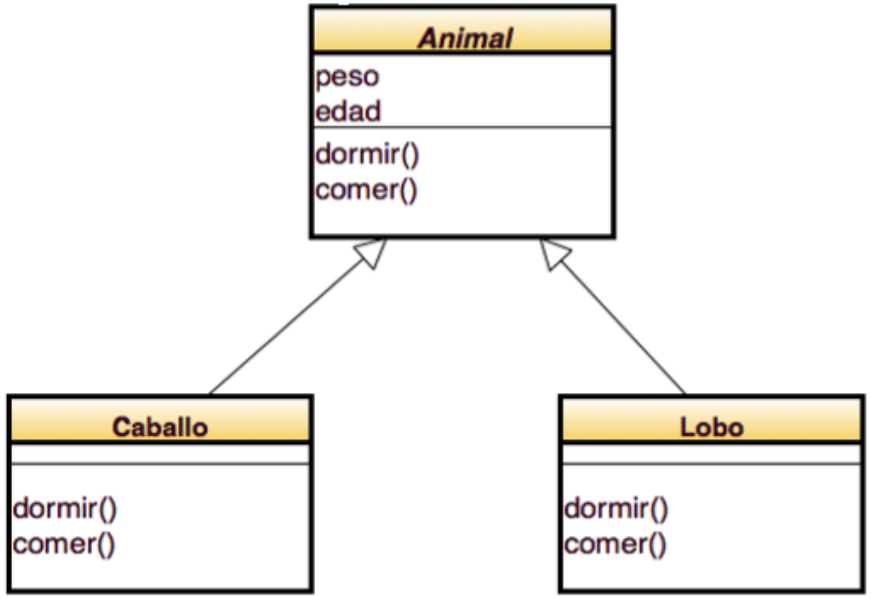
Una clase concreta posee instancias y constituye un modelo completo de objeto (todos los atributos y métodos describen un comportamiento).

Por el contrario, una clase abstracta no puede poseer una instancia directa ya que no proporciona una descripción completa. Su finalidad es poseer subclases concretas y sirve para factorizar los atributos y métodos comunes a las subclases.

En UML, las clases o métodos abstractos se representan mediante el estereotipo <<abstract>>. Gráficamente se representan explícita, o bien implícitamente, poniendo en cursiva el nombre de la clase o método.

Clases concretas y abstractas

Los animales pueden dormir o comer, pero lo hacen de manera distinta de acuerdo con la naturaleza del animal. La clase Animal será abstracta tanto la clase como sus propios métodos. La clase abstracta Animal presenta la redefinición para hacerlos concretos, es decir, los caballos duermen de pie mientras que los lobos duermen tumbados. Tampoco comen de la misma manera: los lobos son carnívoros y los caballos son herbívoros.

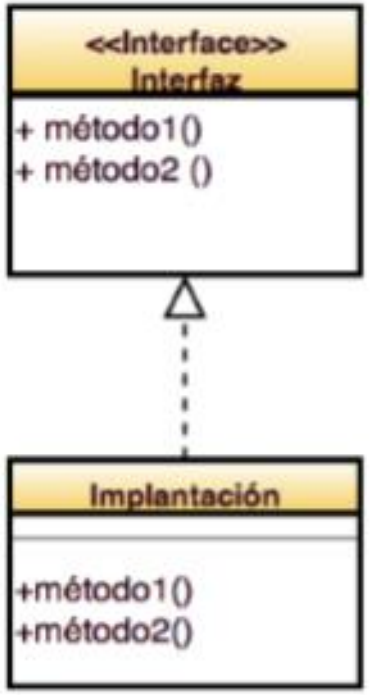


Interfaz

Una interfaz es una clase totalmente abstracta, es decir, no tiene atributos y todos sus métodos son abstractos y públicos. Gráficamente se representa con el estereotipo <<interface>>.

La implantación de los métodos se realiza mediante una o varias clases concretas, subclases de la interfaz. En este caso, la relación de herencia entre la interfaz y la subclase de implantación se conoce como relación de realización. En las relaciones de herencia entre dos clases, se representa gráficamente mediante una línea discontinua en lugar de una línea completa.

Interfaz



Una interfaz es una **colección de operaciones que representan servicios ofrecidos por una clase o componente**.
Por definición todas las operaciones tendrán una visibilidad pública.