UD 06.DIAGRAMAS DE ESTRUCTURA. DIAGRAMAS DE CLASES

ceedcv
CENTRE ESPECÍFIC D´EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Entornos de desarrollo (ED)

Sergio Badal
Raúl Palao

Extraído de los apuntes de:
Cristina Álvarez Villanueva; Fco. Javier Valero Garzón; M.ª Carmen Safont; Paco Aldarias

UD 06.DIAGRAMAS DE ESTRUCTURA

6 DIAGRAMAS DE ESTRUCTURA

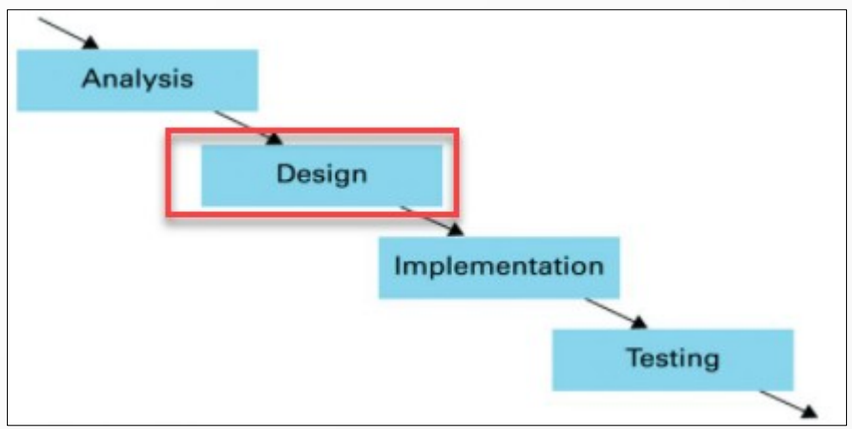
6.1 DISEÑO ORIENTADO A OBJETOS

6.2 DIAGRAMAS DE ESTRUCTURA

6.3 DIAGRAMAS DE CLASES

6.1 DISEÑO ORIENTADO A OBJETOS

- **DISEÑO DE SOFTWARE ORIENTADO A OBJETOS**
  - Dejamos atrás la fase de ANÁLISIS (casos de uso) y entramos en fase de DISEÑO DE SOFTWARE.
  - Concretamente, trataremos el **DISEÑO DE SOFTWARE desde el punto de vista ORIENTADO A OBJETOS (OO)** (clases, métodos…) desde una visión conceptual (teórica).
  - No veremos, por tanto, PROGRAMACIÓN ORIENTADA A OBJETOS pero sí que **haremos referencias puntuales a ella, concretamente a Java**.



UD 06.DIAGRAMAS DE ESTRUCTURA

6 DIAGRAMAS DE ESTRUCTURA

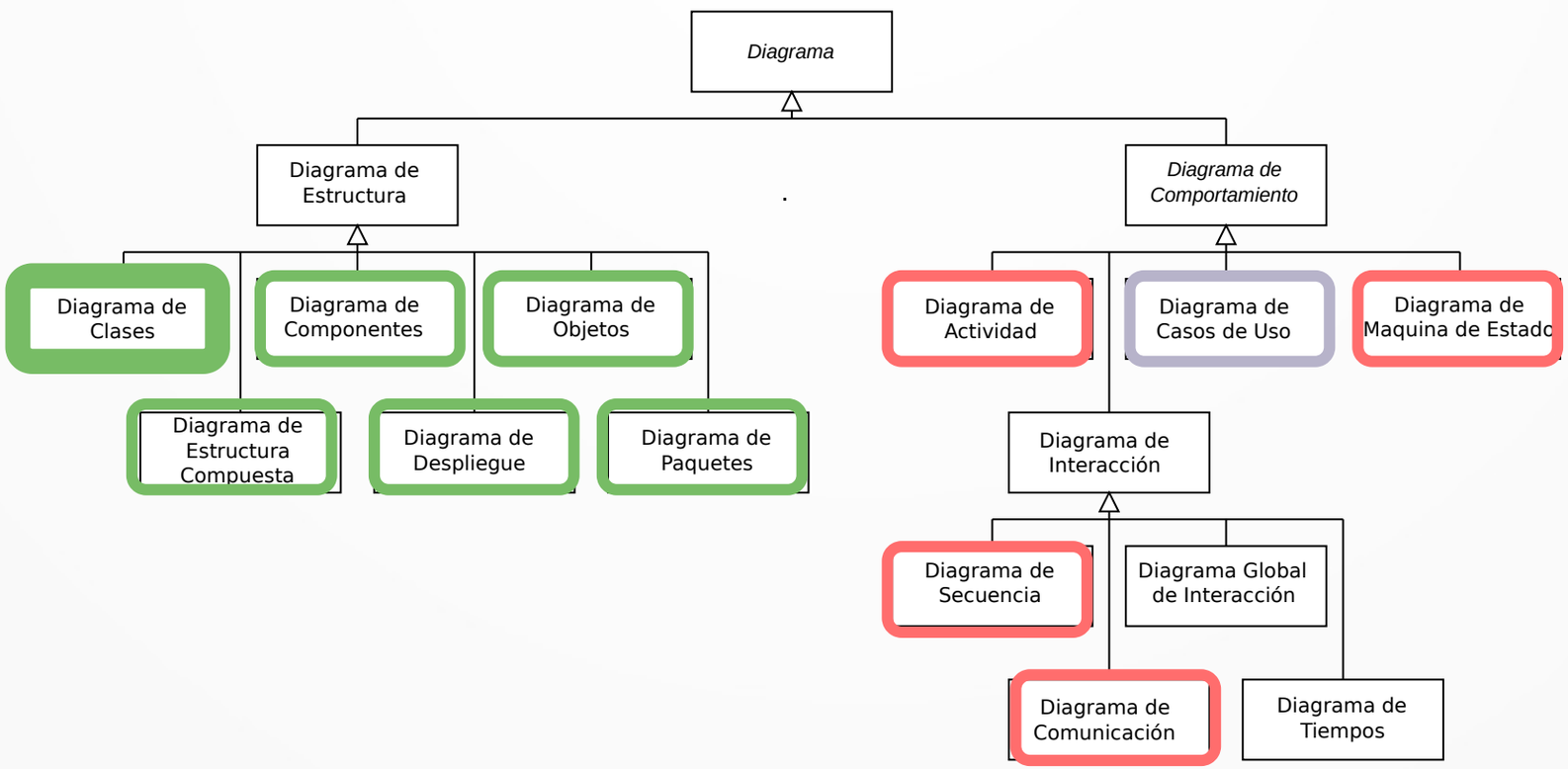
6.1 DISEÑO ORIENTADO A OBJETOS

6.2 DIAGRAMAS DE ESTRUCTURA

6.3 DIAGRAMAS DE CLASES

6.2 DIAGRAMAS DE ESTRUCTURA

En su última versión, UML 2.5.1 de 2015, tiene 13 diagramas:



6.2 DIAGRAMAS DE ESTRUCTURA

- **DIAGRAMAS DE ESTRUCTURA**
  - Los diagramas estructurales existen para visualizar, especificar, construir y documentar los aspectos estáticos del sistema. Los diagramas estructurales incluyen:
    - **Diagramas de Clases**
    - Diagramas de Objetos
    - Diagramas de Despliegue
    - Diagramas de Componentes

UD 06.DIAGRAMAS DE ESTRUCTURA

6 DIAGRAMAS DE ESTRUCTURA

6.1 DISEÑO ORIENTADO A OBJETOS

6.2 DIAGRAMAS DE ESTRUCTURA

6.3 DIAGRAMAS DE CLASES

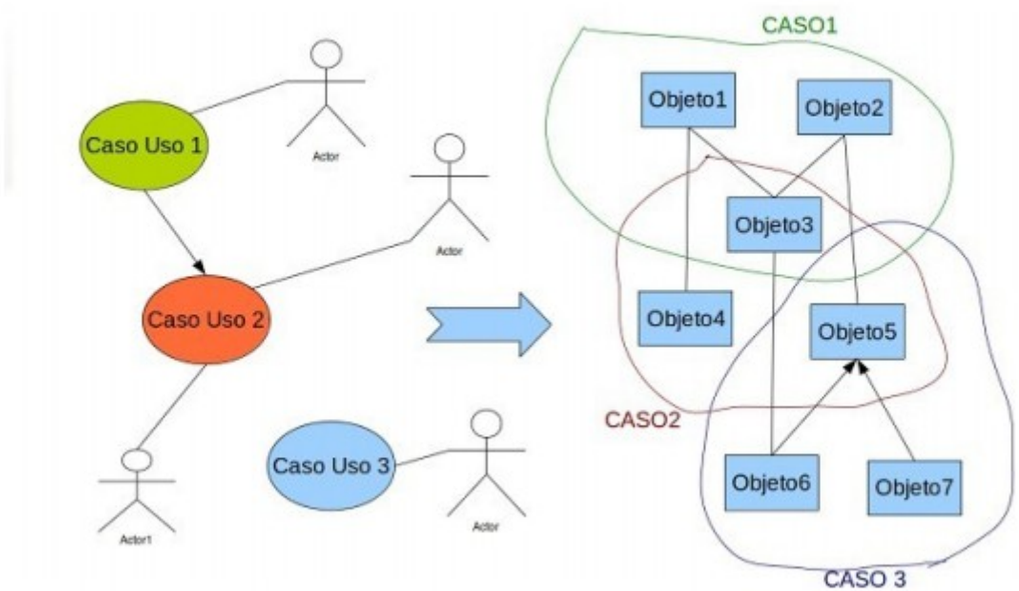
6.3 DIAGRAMAS DE CLASES

DIAGRAMA DE CLASES

DE LOS CASOS DE USO (ANÁLISIS)

… A LAS CLASES (DISEÑO)

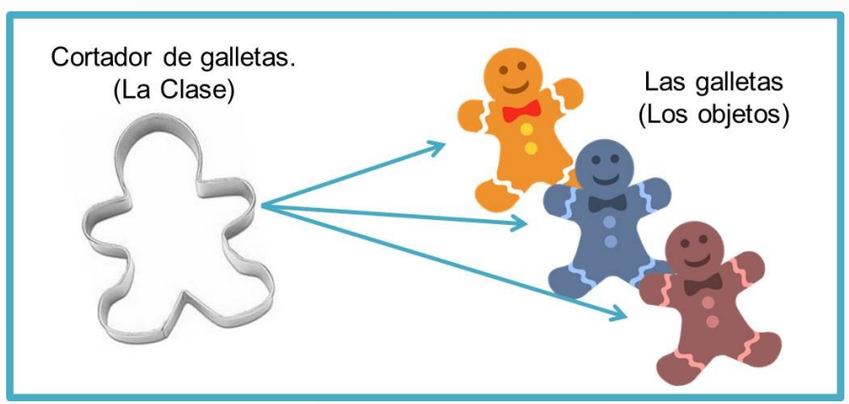
Partiendo de un diagrama de casos de uso (DCU) se puede obtener el diagrama de clases (DC). Cada parte del diagrama de clases será un caso de uso diferente. Esas clases, se instanciarán a objetos los cuales se asocian al diagrama de clases.



6.3 DIAGRAMAS DE CLASES

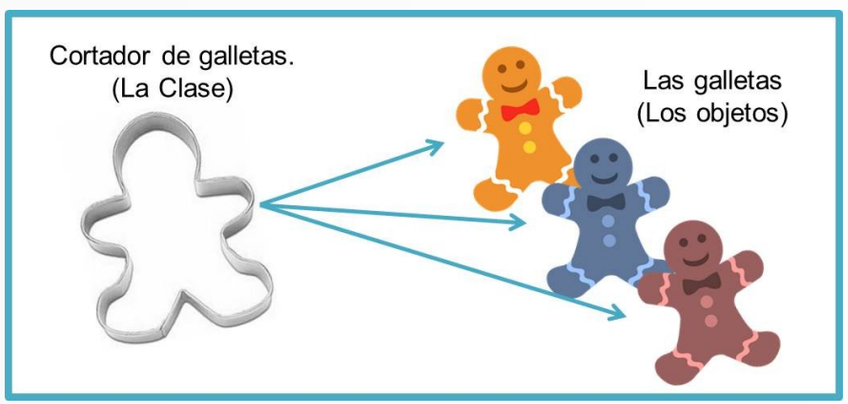
OBJETO VS CLASE

- Un objeto es una **ABSTRACCIÓN**, concepto o cosa con límites bien definidos y con significado en el sistema.
- **Un objeto de una determinada clase se denomina una instancia de la clase.**
- Estos objetos interactúan unos con otros.
  - Cada objeto es capaz de recibir mensajes, procesar datos y enviar mensajes a otros objetos de manera similar a un servicio.
- **Los conceptos de clase y objetos son análogos a los de tipo de datos y variable.**
  - Definida una clase podemos crear objetos de esa clase.
  - Igual que disponiendo de un determinado tipo de dato (por ejemplo el tipo entero), podemos definir variables de dicho tipo.



6.3 DIAGRAMAS DE CLASES

OBJETO VS CLASE



- **Una clase es una descripción de un conjunto de objetos con las mismas propiedades (atributos), el mismo comportamiento (funciones), las mismas relaciones con otros objetos.**
  - Es una plantilla para crear objetos. En POO, una clase es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo.
- **El DIAGRAMA DE CLASES describe el estado y el comportamiento que todos los objetos de la clase comparten.**

6.3 DIAGRAMAS DE CLASES

DIAGRAMA DE CLASES

El diagrama de clases –en adelante, DC- modela la vista estática del sistema, ya que no describe el comportamiento del sistema en función del tiempo. Sus elementos principales son:

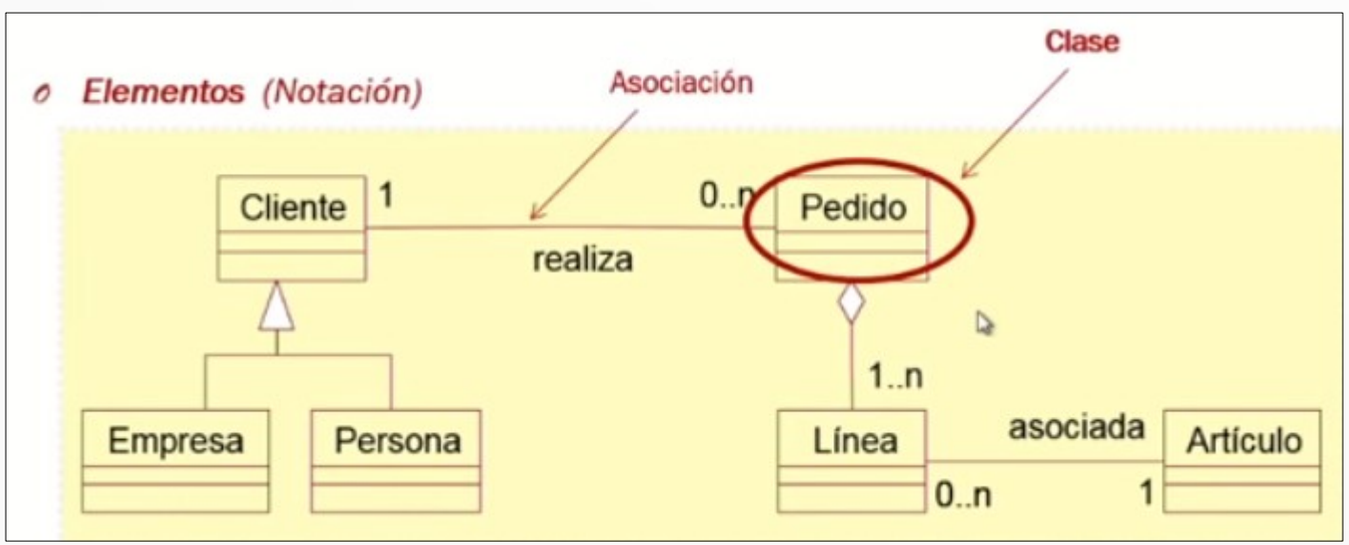
- **Clases**: para modelar un concepto del dominio de la aplicación o solución.
- **Relaciones**: para asociar clases.

Los DCs son los diagramas más comunes en el modelado de sistemas orientados a objetos.

Aquellos que incluyen clases activas se utilizan para cubrir la vista de procesos estática de un sistema.

Además, pueden incluir:

- La especificación o **conceptualización**: recoge los conceptos del problema. No representa una definición para el software.
- El diseño o **implementación**: aquello que forma parte de la programación.



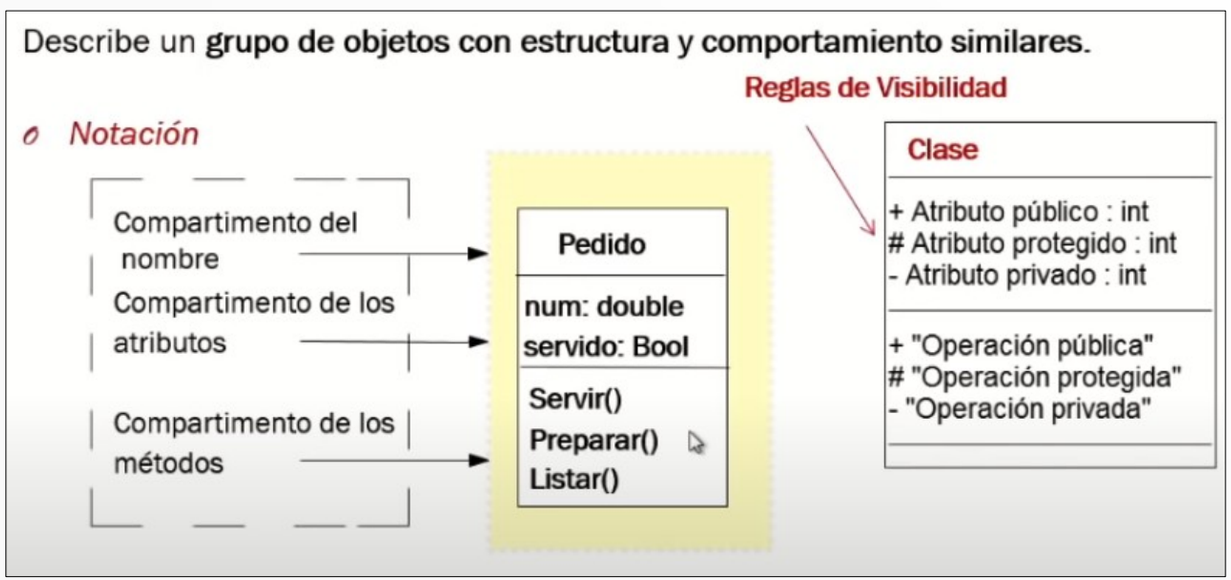
Fuente: Diagrama de clases UPV @ https://www.youtube.com/watch?v=JioEGJIlg88

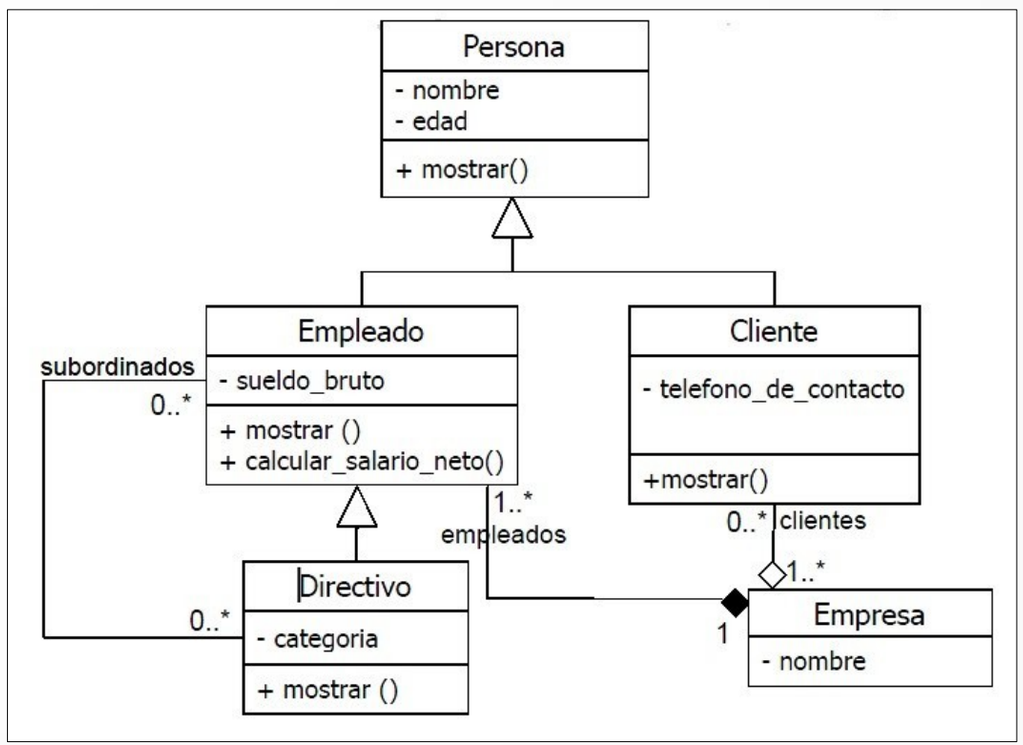
6.3 DIAGRAMAS DE CLASES

CONCEPTO DE CLASE

Describe un **grupo de objetos con estructura y comportamiento similares.**

- *Notación*

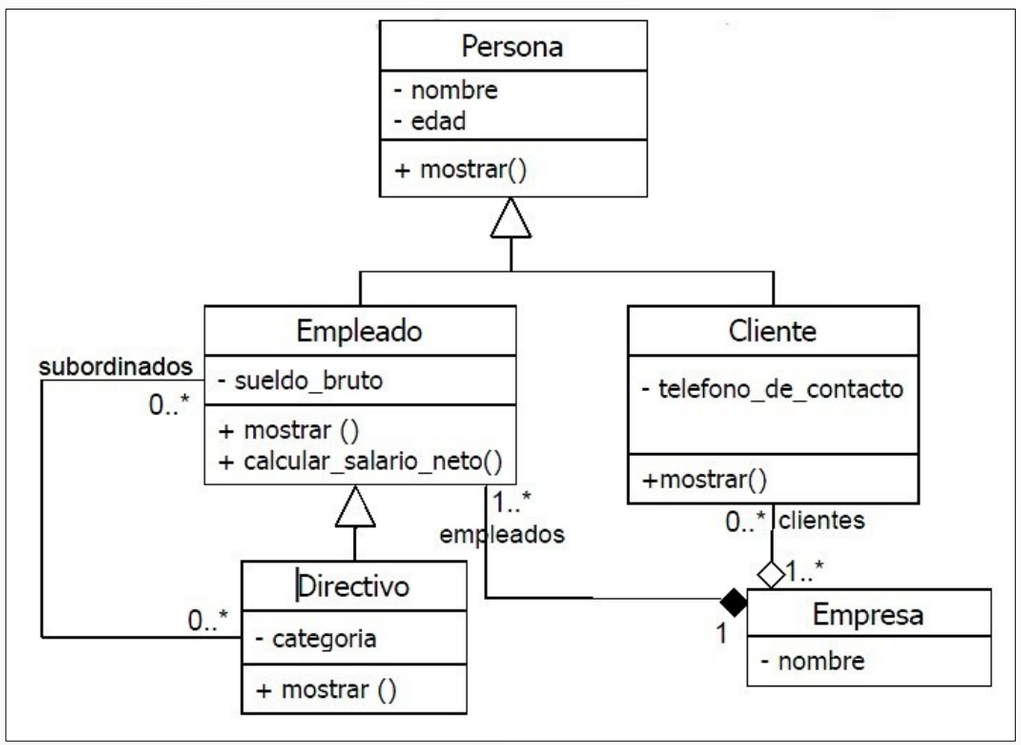




Fuente: Diagrama de clases UPV @ https://www.youtube.com/watch?v=JioEGJIlg88

6.3 DIAGRAMAS DE CLASES

CONCEPTO DE CLASE



Fuente de la imagen: https://gabrielaesteba.wordpress.com/2017/04/11/programacion-orientada-a-objetos/

6.3 DIAGRAMAS DE CLASES

VISIBILIDAD DE UNA CLASE/OBJETO

Un objeto siempre puede acceder a cualquier atributo o método de la clase a la que pertenece y usarlo sin restricción. Para poder establecer la visibilidad de los atributos y métodos, se definen los niveles de acceso de los atributos y operaciones, que son:

- **Público:**
  - Se representa con el signo **+**.
  - Cualquier clase puede acceder a cualquier atributo o método declarado como público.
  - Si todo elemento que pueda ver a la clase, puede ver también el atributo u operación de la clase.
- **Protegido:**
  - Se representa mediante el signo **#**.
  - Cualquier clase hija puede acceder a cualquier atributo o método declarado como protegido en su clase madre.
  - Si sólo pueden ver el atributo u operación indicados los elementos pertenecientes a la clase o a un descendiente de su clase.
- **Privado:**
  - Se representa mediante el signo **-**.
  - Ninguna clase puede acceder a un atributo o método declarado como privado.
  - Si sólo pueden ver el atributo u operación los elementos pertenecientes a esa misma clase.

```
class Alumno {
        // atributos
        private String nombre ;
        private String apellido ;
        private int edad ;

        // métodos de acceso
        public void setNombre ( String pNombre ) {
             nombre = pNombre ;
        }

        public void setApellido ( String pApellido ) {
            apellido = pApellido ;
        }

        public void setEdad ( int pEdad ) {
            edad = pEdad ;
        }

        public String toString () {
            return ( nombre + " tiene " + edad + " anyos ."); 
        }
}
```

[Alumno.java]

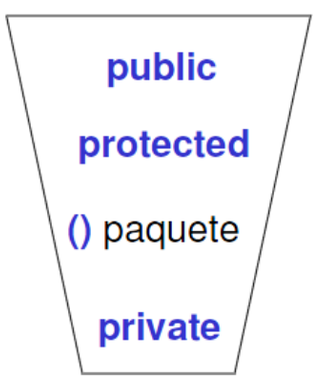
- **Falta una, ¡la visibilidad de paquete!**

6.3 DIAGRAMAS DE CLASES

VISIBILIDAD DE UNA CLASE/OBJETO

Niveles de visibilidad

- En Java, los **niveles de visibilidad son incrementales**



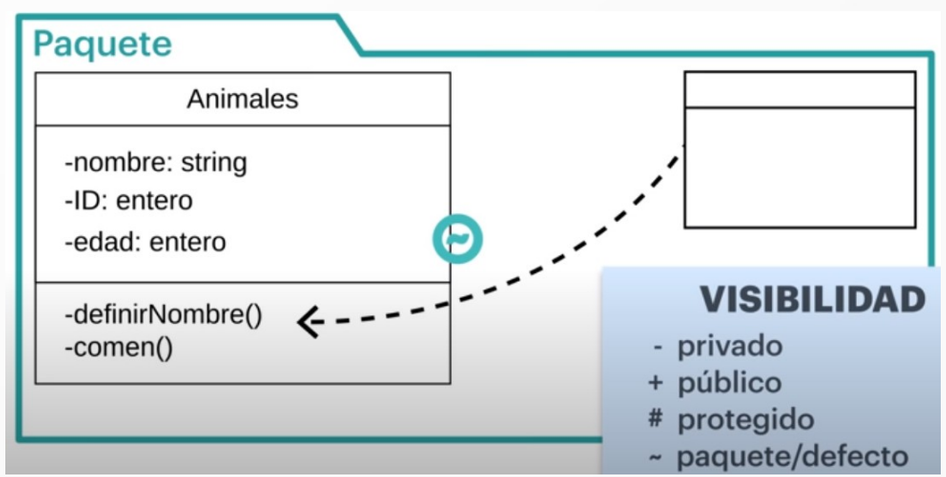
public → todo el código

protected → clase + paquete + subclases

(*nada*) → clase + paquete

private → clase

		Mismo paquete		Otro paquete	
		Subclase	Otra	Subclase	Otra
-	**private**	*no*	*no*	*no*	*no*
#	**protected**	*sí*	*sí*	*sí*	*no*
+	**public**	*sí*	*sí*	*sí*	*sí*
~	*package*	*sí*	*sí*	*no*	*no*

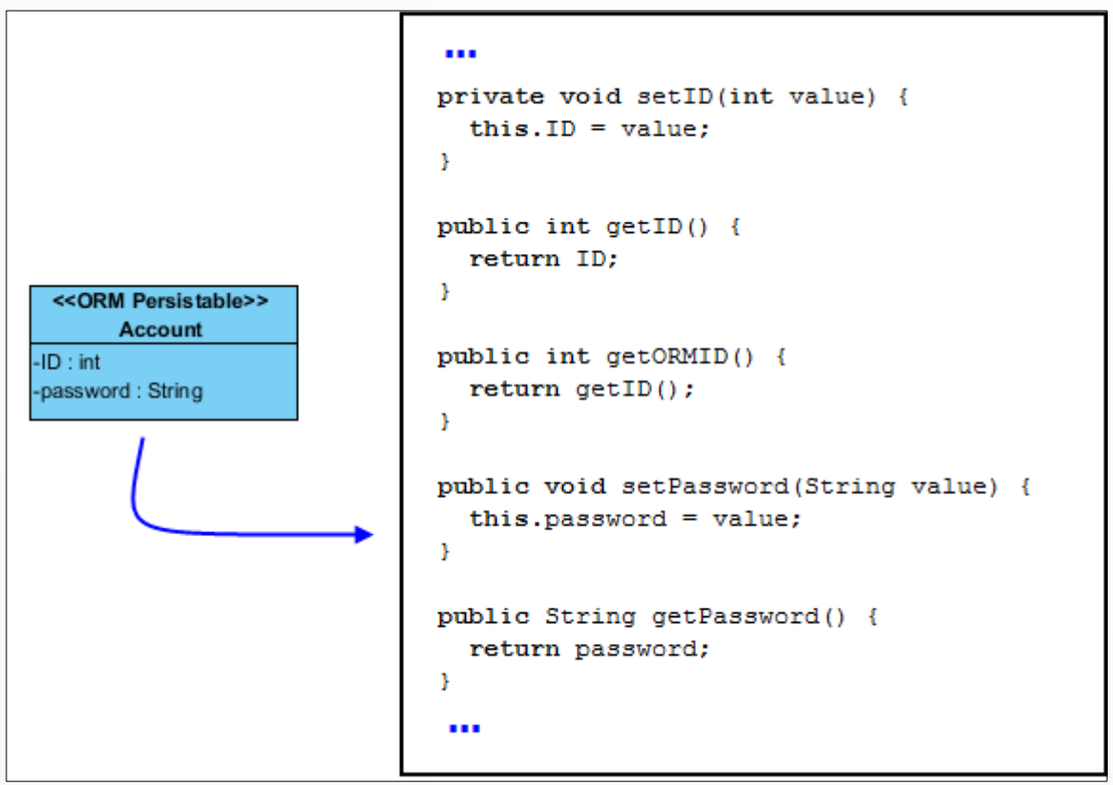


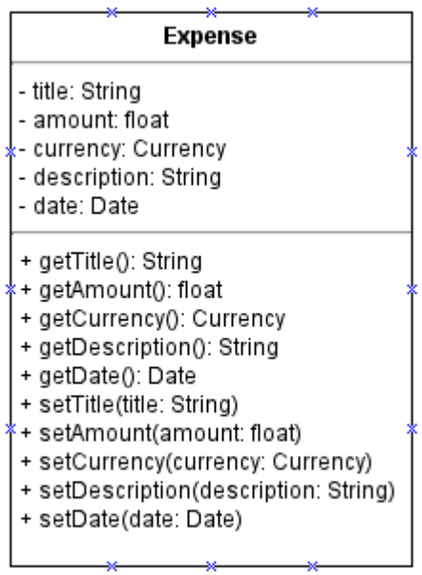
6.3 DIAGRAMAS DE CLASES

SETTERS Y GETTERS

Son métodos que nos sirven para acceder a los atributos de forma segura para leerlos *getters* o modificarlos *setters.*

No suelen incluirse en el diagrama de clases.





6.3 DIAGRAMAS DE CLASES

SETTERS Y GETTERS

Should I include getters & setters in class diagram?

Asked 9 years, 11 months ago Active 7 months ago Viewed 23k times

25

I am required to draw a class diagram for my JSF application for a project documentation. So I have lots of classes as managed beans, with many attributes therefore many getters and setters.

When I draw the class diagram should I also include the getters & setters in the diagram or can I simply leave them?

5

java uml getter-setter class-diagram documentation-generation

The Overflow

Podcast 3
GameStop

Sequencir
and open

Featured on M

32

You are correct: there is no need to include the (noise of) "boilerplate" signatures of standard setters and getters in a class model. Unfortunately, UML does not define a standard notation for implying getters and setters for private attributes. So, you'd have to use your own convention. For instance, you could include a general explanation (that all private properties have getters and setters, while private read-only properties have only getters) as a UML *Comment*, shown as a rectangle with the upper right corner bent (also called a "note symbol") attached to the diagram.

If you prefer to make the getter/setter convention more explicit for the properties concerned, then create your own stereotypes (e.g., «get/set» and «get») to be used for categorizing these private properties, as shown in the following diagram:

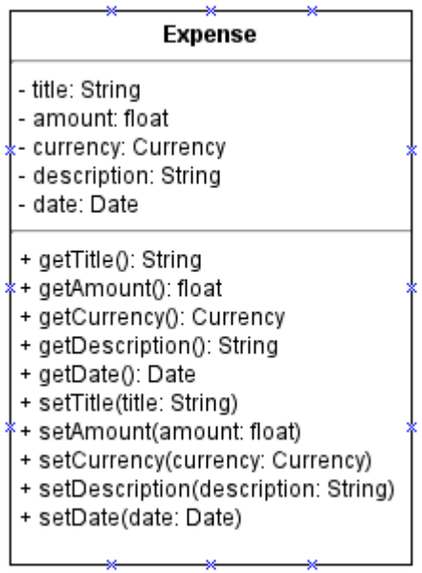
Expense
«get/set» -title : String «get/set» -amount : float «get/set» -date : Date «get» -recordCreatedOn : TimeStamp
+doSomething()

https://stackoverflow.com/questions/28139621/shortcut-for-denoting-or-implying-getters-and-setters-in-uml-class-diagrams/28141950

6.3 DIAGRAMAS DE CLASES

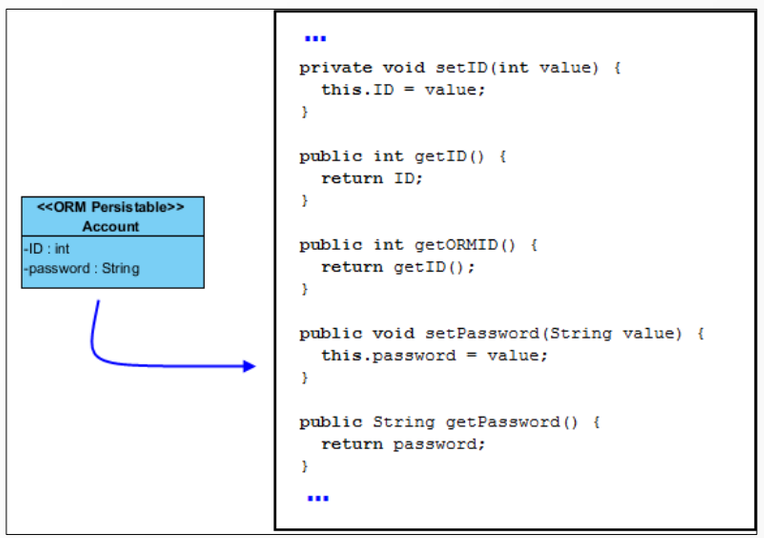
SETTERS Y GETTERS

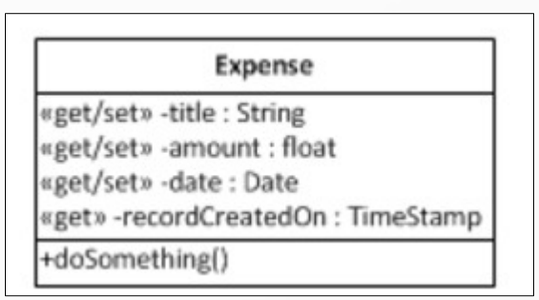
Si no te piden expresamente que los incluyas en el diagrama de clases, tienes tres opciones:



a) Ponerlos

b) No ponerlos





c) Incluirlos de forma implícita

6.3 DIAGRAMAS DE CLASES

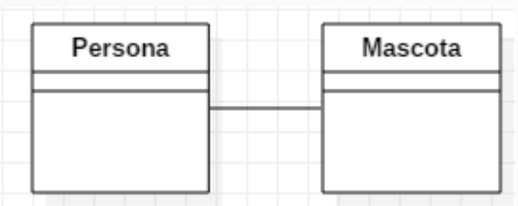
RELACIÓN ENTRE CLASES

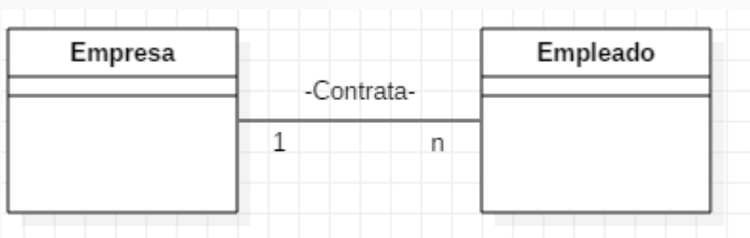
Se puede definir una relación como una abstracción de un conjunto de interrelaciones semánticas puntuales que se dan sistemáticamente entre diferentes tipos de objetos.

Las relaciones pueden ser de distintos tipos:

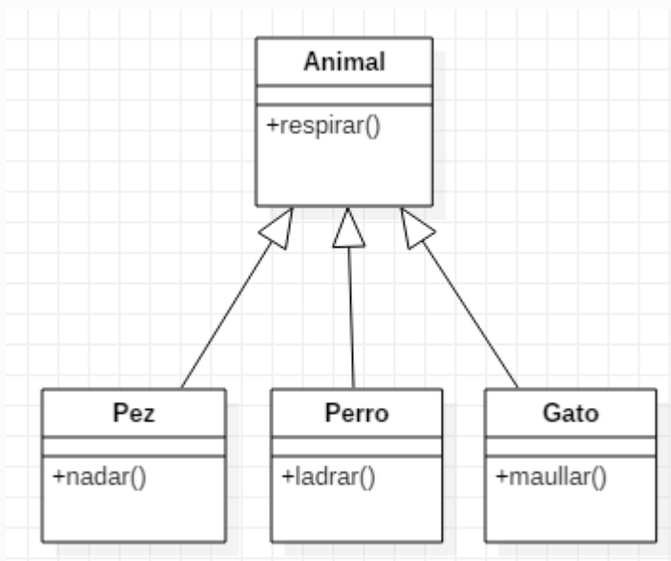
- Asociación
- Herencia (Generalización/Especialización).
- Agregación/Composición.

A) ASOCIACIÓN

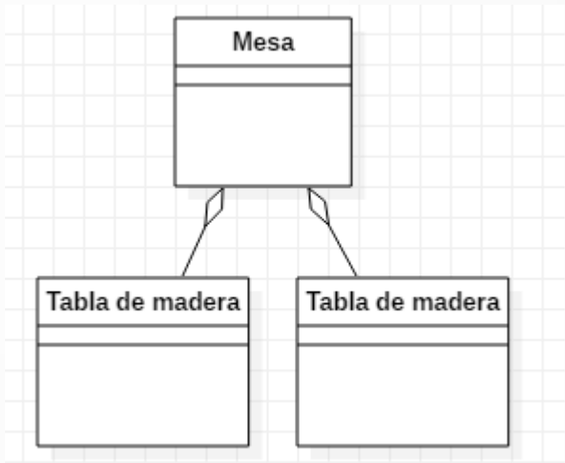




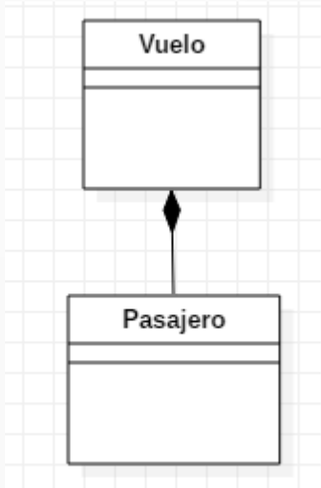
B) HERENCIA



C) AGREGACIÓN



D) COMPOSICIÓN



6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES

A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

La asociación representa relaciones entre objetos, sin ningún carácter especial. Constituye una relación semántica bidireccional entre clases.

Una asociación es una relación estructural que describe una conexión entre objetos.

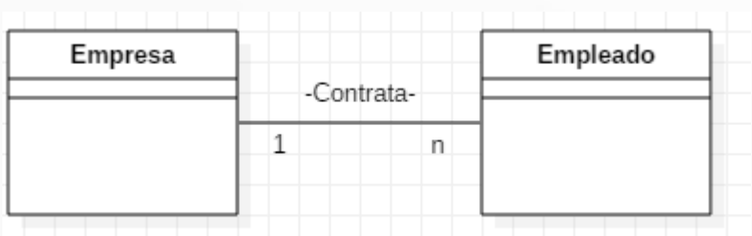
Gráficamente, se muestra como una línea continua que une las clases relacionadas entre sí.

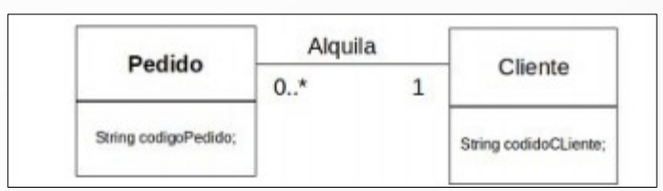
La asociación no es un flujo de datos sino un enlace entre los objetos de las clases asociadas.

La asociación debe ser nombrada mediante un verbo.

Pueden ser:

- Unaria: entre la misma clase
- Binaria: entre dos clases
- Ternaria: entre tres clases
- N-aria: entre tres o más clases





En código sería algo como:

```
empresaA = new Empresa(..);
...
empleadoX = new Empleado(..);
...
empresaA.contrata(empleadoX);
```

En código sería algo como:

```
clienteA = new Cliente(..);
...
pedidoX = new Pedido(..);
...
clienteA.alquila(pedidoX);
```

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES

A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

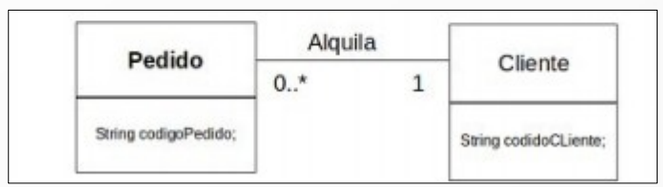
MULTIPLICIDAD

La multiplicidad o cardinalidad define cuántas instancias de un tipo A pueden asociarse a una instancia del tipo B en determinado momento.

Es decir, indica cuántos objetos de una clase se pueden relacionar, con un objeto de la clase asociada. Restringe el número de objetos de una clase que participan en la relación.

MULTIPLICIDAD	SIGNIFICADO
1	Una instancia de cliente debe ser unido con exactamente una instancia de película no más y no menos.
*	Una instancia puede ser unida con cero o más instancias de película.
0..*	Una instancia de cliente puede ser unida con cero o más instancias de película. Es equivalente al anterior.
0..1	Una instancia de cliente debe ser unida con cero o una instancia de película. Esto se llama multiplicidad opcional.
1..*	Una instancia de Cliente debe ser unido con al menos una o más intancias de película.
5..9	Una instancia de cliente puede ser unida con al menos cinco instancias de película pero no más de 9.
3,5,7	Una instancia de cliente puede ser unido a 3, o 5 o 7 instancias de película.

Asociación:
1 cliente realiza mínimo 0 y máximo N pedidos
1 pedido es realizado como minimo/maximo 1 cliente



En código sería algo como:

```
clienteA = new Cliente(..);
...
pedidoX = new Pedido(..);
...
clienteA.alquila(pedidoX);
```

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES

A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

MULTIPLICIDAD

UNIFIED MODELING LANGUAGE

	Una *A* siempre se asocia con una *B*	Una *A* siempre se asocia con una o más *B*	Una *A* siempre se asocia con ninguna o con una *B*	Una *A* siempre se asocia con ninguna, con una o con más *B*
Booch (1ª ed.)	A —1— B	A —+— B	A —?— B	A —*— B
*Booch (2ª ed.)**	A —1 *— B	A —1..N— B	A —0..1— B	A —N— B
Coad	A —1 — B	A —1,m— B	A —0,1— B	A —0,m— B
*Jacobson***	A —[1]→ B	A —[1,M]— B	A —[0,1]→ B	A —[0,M]→ B
Martin/ Odell	A —‖— B	A —⋔— B	A —○|— B	A —○⋔— B
Shlaer/ Mellor	A —→ B	A —→ B	A —C→ B	A —C→ B
Rumbaugh	A — B	A —1+● B	A —○— B	A —●— B
Unified	A —1— B	A —1..*— B	A —0..1— B	A —*— B

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

RESTRICCIONES

Un diagrama de clases a través de dibujo está expresando restricciones que deben cumplir las clases.

De este modo la cardinalidad sería una restricción entre clases. Por ejemplo un pedido sólo puede ser asignado a un cliente.

Pero hay veces que el diagrama no expresa todos los requisitos, para ello se utilizan las restricciones de integridad.

UML no define una sintaxis estricta para describir las restricciones, aparte de ponerlas entre llaves ({}). Se puede escribir en lenguaje informal, para simplificar su lectura. Lo idea sería escribir un fragmento de código de programación, si ya sabemos con qué lenguaje se va a programar.

Ejemplo:

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES

A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

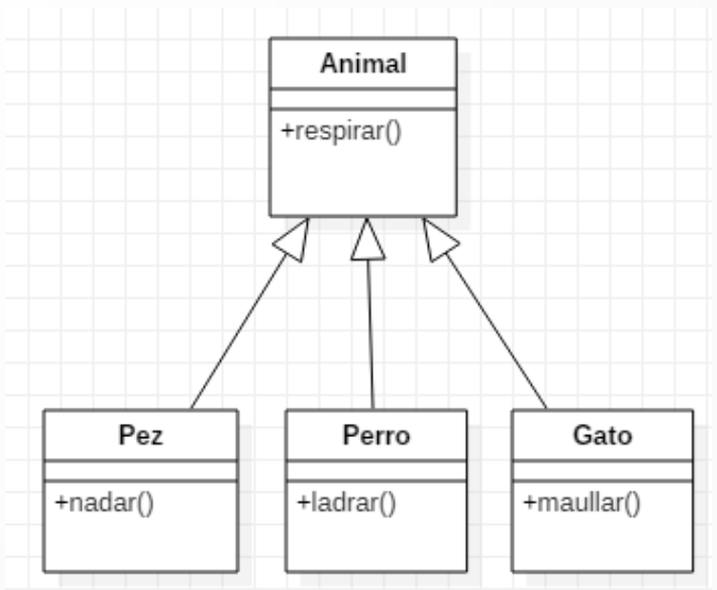
La herencia define la relación entre una superclase y una o varias subclases en la que ambas, recogen el mismo concepto, pero las subclases con un nivel de detalle mayor que las superclases.

Las subclases podrán tener más de una superclase, lo que se denomina herencia múltiple.

Una subclase puede ser ampliada con atributos y operaciones adicionales que se aplican únicamente a ese nivel de jerarquía, y puede proporcionar su propia implementación de una operación heredada.

La herencia se puede encontrar mediante dos métodos:

- **Generalización**: proporciona el mecanismo para crear superclases que encapsulan atributos y comportamiento común de varias clases.
- **Especialización**: proporciona la capacidad para crear subclases que representan el mismo concepto con un nivel mayor de refinamiento y detalle de la superclases.



En código sería algo como:

```
public class Animal{
…
{
```

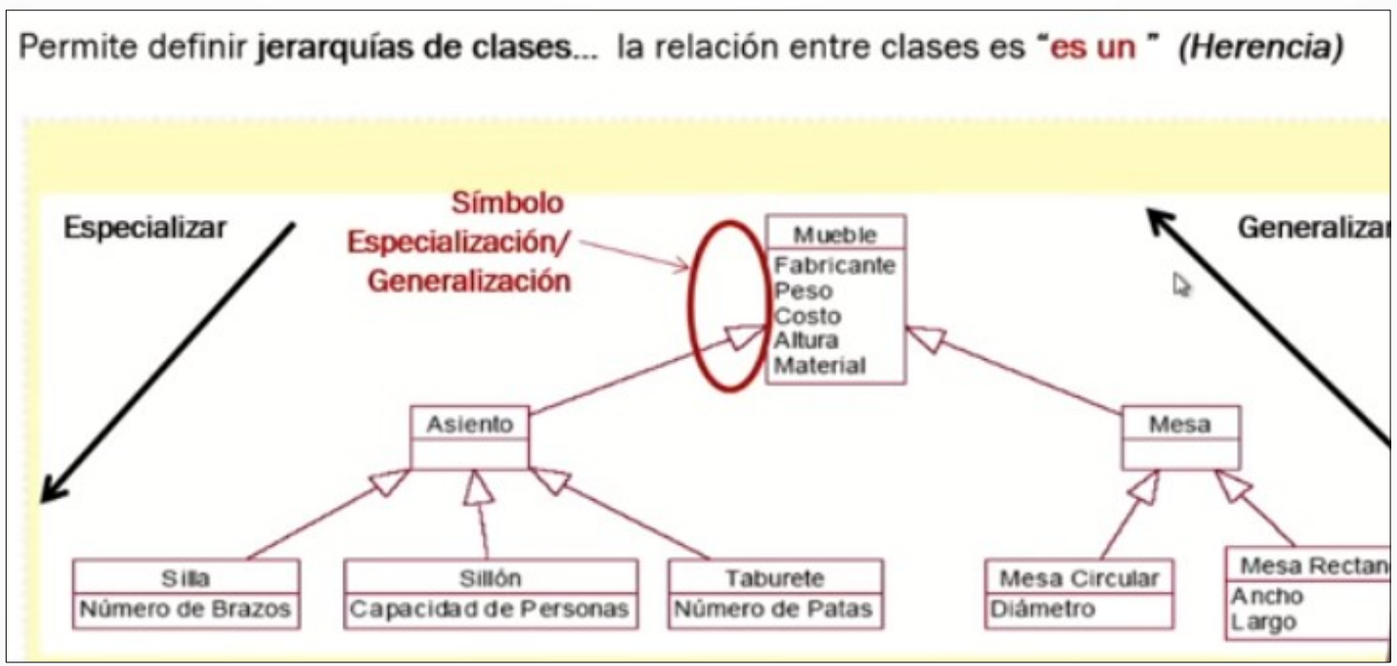
```
public class Pez extends Animal{
…
{
```

```
public class Perro extends Animal{
…
{
```

```
public class Gato extends Animal{
…
{
```

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

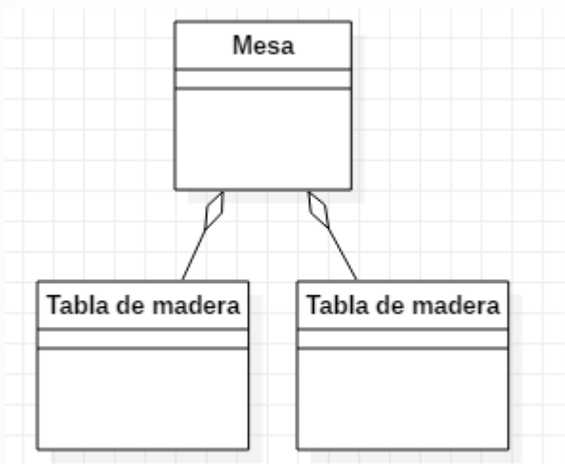


Fuente: Diagrama de clases UPV @ https://www.youtube.com/watch?v=JioEGJIlg88

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

La agregación es un tipo de asociación que recoge una semántica especial. Son relaciones en las que un objeto está compuesto por otros del mismo o de diferentes tipos, es decir, relaciones del todo-parte.

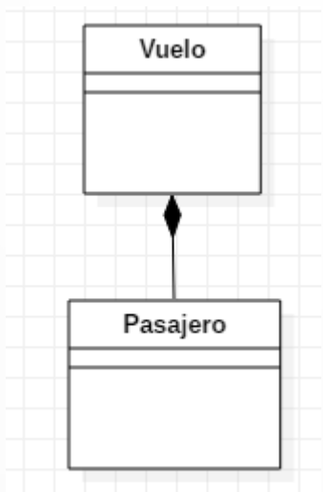


La **agregación** es una forma especial de asociación que relaciona Todo/Parte:

```
public class Todo {
   private Parte esParte;
}
```

6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN



La **composición** es una forma especial de agregación. Su implementación en java es indistinguible de la asociación. Esto es debido a que la composición no se usa mucho en java.

```
public class Propietario {
   private Sala esSala;
}
```

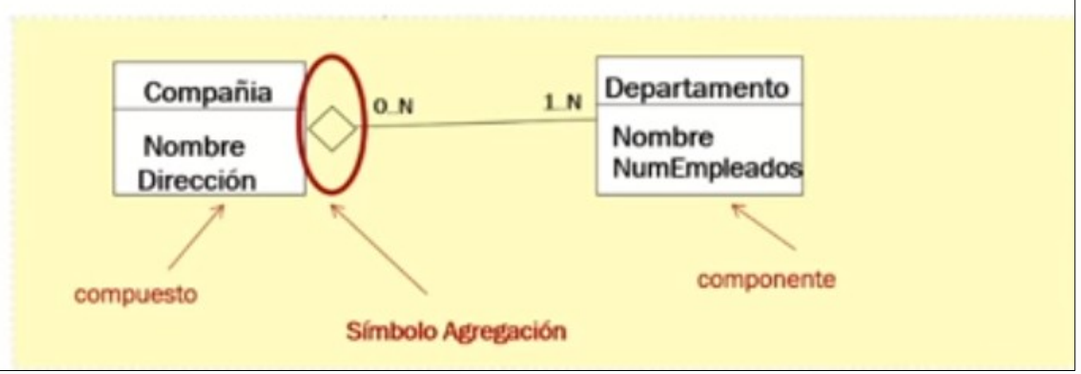
6.3 DIAGRAMAS DE CLASES

IMPORTANTE

RELACIÓN ENTRE CLASES A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

Agregación

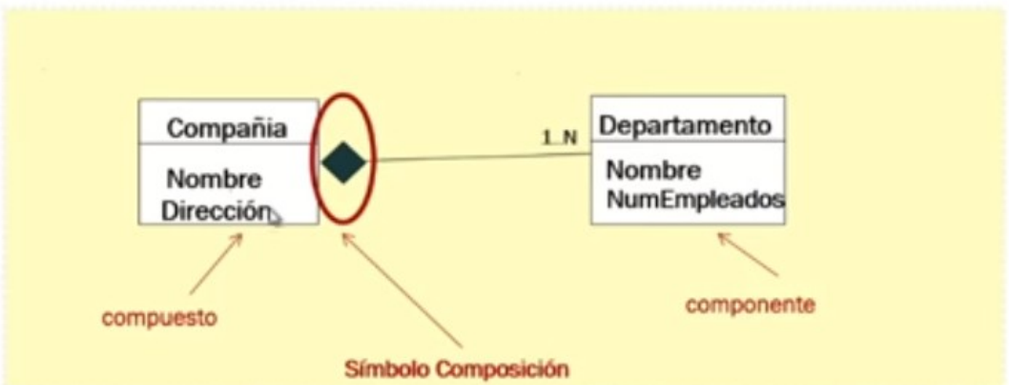
Es una **asociación** con más semántica.... la relación entre clases es "está formado po



- En el caso de la izquierda, un mismo departamento puede pertenecer (o estar asociado) a varias compañías.
- Pensad en una empresa de servicios, cuyos departamentos son "redacción de textos", "redes sociales".. y que trabajan para otras compañías.

Composición

Es una **agregación inclusiva** .... la relación entre clases es "está compuesto por"



- En el caso de la derecha una compañía tiene varios departamentos.
  - Si elimino la empresa, elimino sus departamentos.
- Dos diagramas con las mismas clases, tienen relaciones distintas dependiendo del contexto.

Ambas opciones son válidas.

¡Dos diagramas con las mismas clases, tienen relaciones distintas dependiendo del contexto!

Fuente: Diagrama de clases UPV @ https://www.youtube.com/watch?v=JioEGJIlg88

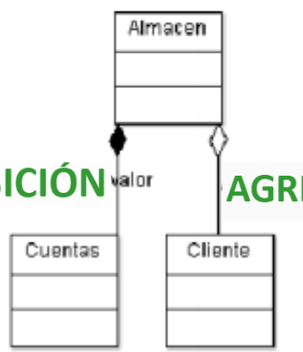
6.3 DIAGRAMAS DE CLASES

RELACIÓN ENTRE CLASES

A) ASOCIACIÓN B) HERENCIA C) AGREGACIÓN D) COMPOSICIÓN

Veamos más ejemplos.

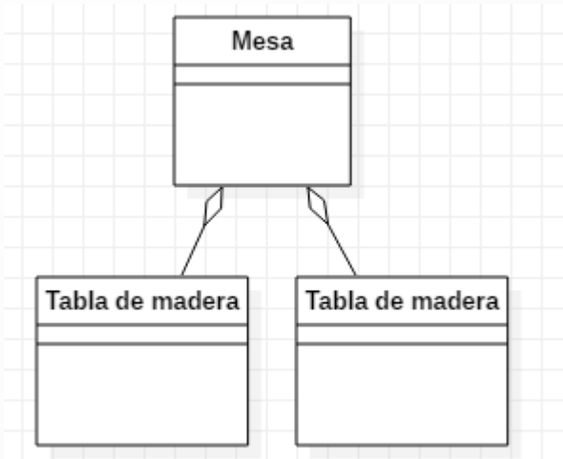
Ejemplo- Almacén: "Un Almacén posee Clientes y Cuentas. Las cuentas siempre van asociadas a los almacenes. Y los clientes pueden estar asociados a 0 o muchos almacenes"

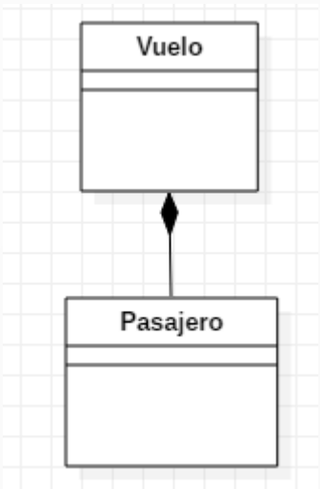


COMPOSICIÓN

AGREGACIÓN

Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias). Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados. La composición (por Valor) se destaca por un rombo relleno. La agregación (por Referencia) se destaca por un rombo transparente.





Ejemplo- Aeropuerto-Ciudad:

Ciudad tiene una lista de objetos de tipo aeropuerto, esto quiere decir, que una ciudad, tiene un número de aeropuertos. Una ciudad puede no tener aeropuertos. Se trata de una relación de agregación

Ejemplo Avión-Ala:

Se trata de un ejemplo de Composición. Un avión siempre está compuesto de dos alas. Deben existir siempre las alas.

AGREGACIÓN COMPOSICIÓN

6.3 DIAGRAMAS DE CLASES

CLASES ABSTRACTAS

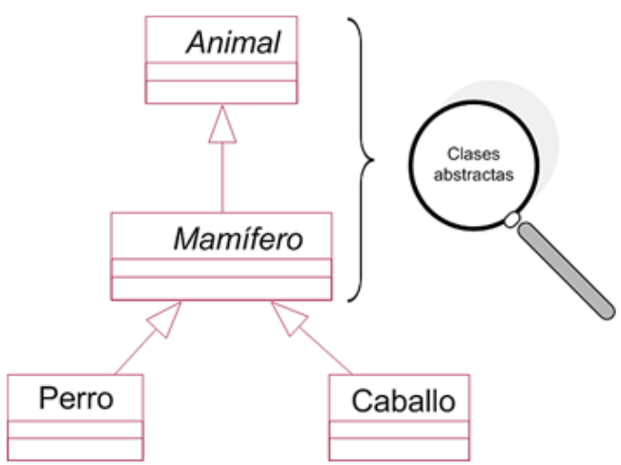
Si examinamos la jerarquía presentada en la figura 3.4 vemos que en ella existen dos tipos de clases:

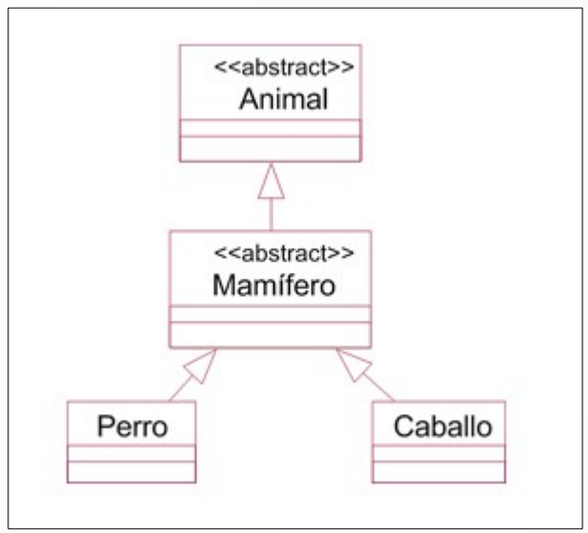
- Las clases que poseen instancias, es decir, las clases `Caballo` y `Perro`, llamadas clases concretas.
- Las clases que no poseen directamente instancias, como la clase `Animal`. En efecto, si bien en el mundo real existen caballos, perros, etc., el concepto de animal propiamente dicho continuá siendo abstracto. No basta para definir completamente un animal. La clase `Animal` se llama clase abstracta.

La finalidad de las clases abstractas es poseer subclases concretas y sirven para factorizar atributos y métodos comunes a las subclases.

Ejemplo

La figura 3.6 retoma la jerarquía detallando las clases abstractas y las clases concretas. En UML, el nombre de las clases abstractas se escribe en cursiva.





Una clase abstracta se denota con el nombre de la clase y de los métodos con letra en cursiva. Esto indica que la clase definida no puede ser instanciada pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es definiendo subclases, que implementan los métodos abstractos definidos.

La clase abstracta obliga a que haya una herencia.

El concepto de clase abstracta es un concepto especializado del concepto de clase. Hemos visto que una clase abstracta se representa como una clase con un nombre en cursiva. Esta representación gráfica incluye un estereotipo implícito, pero también podemos prescindir de poner el nombre de la clase en cursiva y precisar de forma explícita el estereotipo «`abstract`».

Fuente: https://www.ediciones-eni.com/open/mediabook.aspx?idR=0027b432ac7c6e4b0f24eb4e18329e6c

6.3 DIAGRAMAS DE CLASES

DECÁLOGO DE RECOMENDACIONES

No todos los diagramas que encontrarás seguirán estas recomendaciones ya que son solo eso: recomendaciones.

[clases]

1. Nombra las clases con sustantivos en singular, UpperCamelCase y en cursiva si son abstractas.
2. No incluyas las clases que no representen una entidad del sistema como “main”, “test”

[atributos/campos/propiedades]

3. Nombra los atributos con sustantivos lowerCamelCase y en cursiva si son abstractos.
4. Los atributos de una clase suelen ser privados.
5. Los tipos de datos de los atributos suelen ser opcionales (diseño o implementación).

[métodos/operaciones/funciones]

6. Nombra los métodos con verbos lowerCamelCase y solo en cursiva si son abstractos.
7. No incluyas setters, getters, constructores ni destructores, salvo si te los piden.
8. Los métodos suelen ser públicos y los parámetros opcionales (diseño o implementación).

[relaciones/asociaciones]

9. Marca las asociaciones con un rombo relleno (composición) o vacío (agregación).
10. Etiqueta las asociaciones solo cuando sea necesario, con una, dos o hasta tres etiquetas.

6.3 DIAGRAMAS DE CLASES

ERRORES COMUNES

1) Confundir un diagrama de clases con un diagrama entidad-relación.
   Un diagrama de clases muestra un sistema/aplicación en un momento dado (una foto estática) del sistema, no es una foto de la base de datos (aunque coinciden bastante)
2) No uses setters, getters, constructores ni clases como “main”, “test” o “sistema”
3) Se recomienda que las clases sean UpperCamelCase y las propiedades LowerCamelCase.
   => Sigas o no la recomendación, DEBES de ser coherente.
4) Se recomienda que los atributos de una clase que sean privados salvo los de las superclases, que pueden ser protegidos si queremos accederlos desde las subclases sin necesidad de acudir a setters o getters.
   => Sigas o no la recomendación, DEBES de ser coherente.
5) Se recomienda no usar tildes ni caracteres no alfanuméricos
   => Sigas o no la recomendación, DEBES de ser coherente (o los pones todos o ninguno).
6) Se recomienda omitir TODOS "1" e indicar el "0..*" como "*"
   => Sigas o no la recomendación, DEBES de ser coherente (o pones todos los 1 o ninguno).
7) Las herencias, salvo casos muy raros, no DEBEN llevar cardinalidades (al ser 1 a 1).
8) Las agregaciones salvo casos muy raros, DEBEN tener "*" en el lado del rombo relleno.
9) Las composiciones, salvo casos muy raros, DEBEN tener "1" en el lado del rombo relleno.
10) Una agregación nunca puede ser una composición y viceversa.