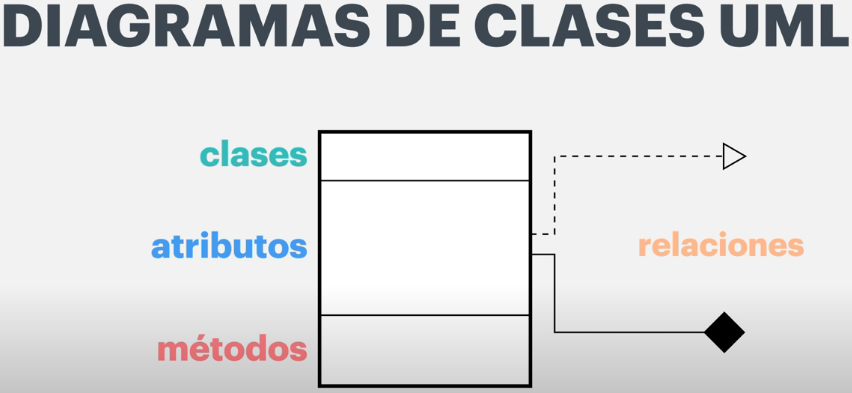
***Diagramas de clases UML***



***Ejemplo de ilustración: Sistema Zoológico***

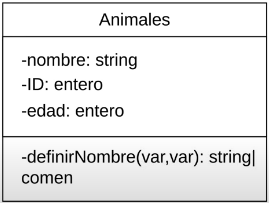
En un zoológico, las “clases” vendrían siendo las diferentes cosas o *protagonistas* (animales, empleados, etc) que hacen parte propiamente del zoológico y hacen que éste **funcione.** Entonces, cada uno de esos elementos; digamos, *empleado domesticador, empleado administrador, animales,...* tienen su propia ***clase.*** Ahora, cada clase, se descompone por una lista de participantes con nombre propio; en el ejemplo de los *animales,* sería un animal en especifico del total de ellos: a esa especificación individual, que se le hace a cada *clase,* de sus elementos se le llama ***instancia.*** Tendríamos entonces... **clase: animal, y sus instancias: león, koala, serpiente, etc.**

Ahora, a la identificación de cada instancia de una clase en particular, por medio de la definición de sus propiedades, se les llama: ***atributos (campos, variables o, bien, propiedades) Ej:***

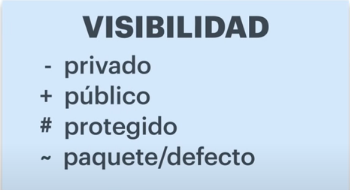
*Clase:* ***animal.*** *Instancia:* ***león.***

*Atributos:* ***nombre del león, ID del léon, edad del león, etc... (a cada atributo se le define con un valor).***

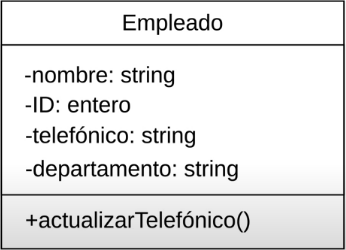
Ahora, vamos a conocer los ***métodos.*** *Los* ***métodos,*** o también funcionesu operaciones, permiten especificar cualquier característica de comportamiento de una clase; es decir, manipular las propiedades o atributos de las instancias de una clase como tal, por medio de ***funciones;*** como, por ejemplo, la función de cambiar o definir el nombre en un animal. Es el lugar donde se declaran las funciones para la clase en cuestión. En la clase animal, por ejemplo, un método sería: comer. Estas *funciones,* por la naturaleza propia de una función (como se conocen en programación), terminan entre parentesis (); y, dentro de esos parentesis, puedes agregar variables (tal como se hace con las funciones en un lenguaje de programación), posteriormente se especifica el tipo de datos, ya sea: “string”, “int”, “char”, etc. Lo que queremos decir con todo esto es que, efectivamente, los métodos o funciones que pongas en esa casillas, se **programan más adelante.**



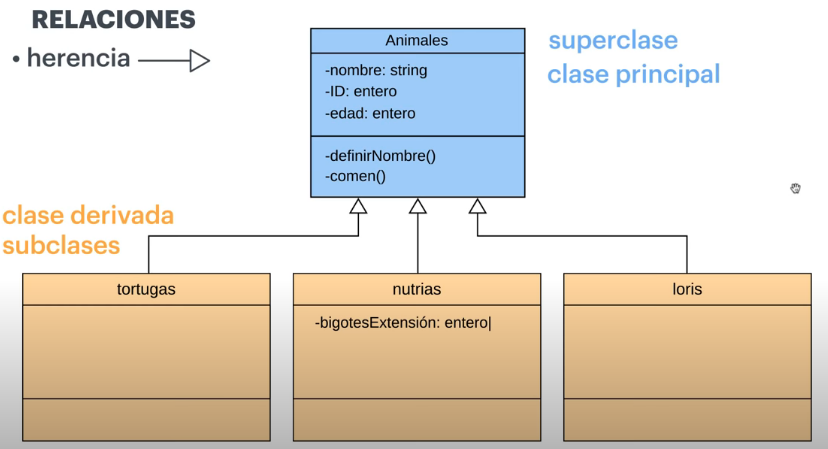
Ahora, hay un concepto nuevo en el recuadro anterior que parece imperceptible: ***la visibilidad,*** la visibilidad nos permite configurar el acceso de los atributos o métodos de una clase. Mejor que lo describa la siguiente imagen:



*En la mayoria de los casos los atributos serán privados o protegidos y los métodos con frecuencia serán públicos. Acá otro ejemplo de clase en zoológico:*

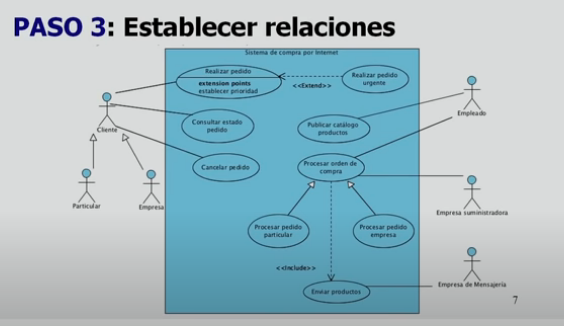


***Relaciones en diagramas de clases UML***



Si se da cuenta, de una clase (con atributas generales para todos), hemos creado sub-clases (donde se pueden especificar atributos todavía más especificos); a esto se le llama: ***herencia***. La *herencia* es una buena práctica que retoma el concepto de *abtracción* para ahorrarnos código y ser más eficientes.

***Tipos de relaciones en UML*** *(pueden ser aplicadas en diagramas de clase o, en lo siguiente que veremos, casos de uso) (las relaciones que vienen después de la herencia)*



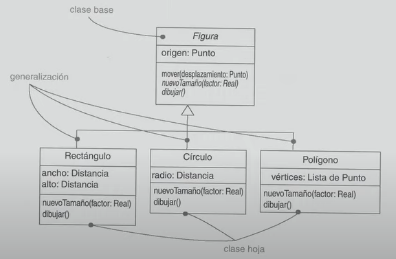
En efecto, las relaciones pueden darse por igual en *diagramas de clase* o en *casos de uso*; de hecho, continuaremos con el pliego de ***relaciones*** restantes, luego de ***herencia,***pero ahora siendo explicadas desde ***casos de uso.***

la flecha con **punta** **negra** se refiere a una: asociación de tipo ***composición*** (es decir, hay una dependencia entre los ambos objetos). Si no se da el caso de uso (proceso) uno, no se da el caso de uso (proceso) dos. Dos (2) es consecuencia de uno (1) y en ese escenario ambos serían importantes en el modelo; en nuestro ejercicio, hay dos casos de asociación de tipo *composición.* **La flecha siempre debe apuntar hacia el caso de uso “padre”, al que no es dependiente.**

Ahora, la flecha con **punta** **blanca** se refiere a una: asociación de tipo ***agregación*** (es decir, NO hay necesariamente una dependencia entre ambos casos de uso). “Sólo se pone de decoración, para *agregar, puntualizar más sobre lo mismo*”. El caso de uso dos (2), acá, es una herramienta para ultimar detalles al caso uno (1), pero no es necesario especificar dentro del modelo. Es decir, dos (2) complementa, le da un valor agregado, a uno (1); pero dos (2), de hecho, no es necesario para uno (1); en nuestro ejercicio, hay dos casos de asociación de tipo *agregación.* **La flecha siempre debe apuntar hacia el caso de uso “padre”, al que no es dependiente.**

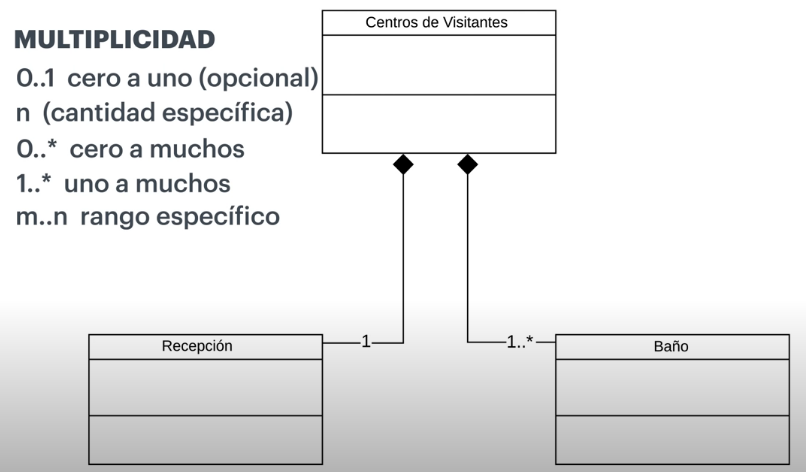
Tanto en ***agregación*** como en ***composición*** se puede dar una asociación de tipo ***generalización,*** que es la misma ***herencia***; es decir, una relación entre elementos conocidos como hijo - padre. El *hijo* tiene la misma especificación del *padre,* la cuál puede extender de forma personalizada en cada *hijo*.

***Ejemplo:***



*Ahora, existe también la relación de la* ***multiplicidad***, que suele darse en una relación de *composición.* En *multiplicidad,* se indica la cantidad de elementos que hay (o los posibles) de un caso de uso *hijo* dentro de un caso de uso *padre;* entendiendo como *hijo-padre* una relación de dependencia. Esto se hace para no tener que repetir el dibujo o diseño de un mismo caso de uso si ha de ser requerido, dicho caso, más de una vez. Simplemente se indica, dentro del mismo caso de uso, la cantidad que necesitas de dicho caso en cuestión (y te ahorras dibujarlos, logrando así una mayor eficiencia).

***Ejemplo:***



***Otras relaciones en UML***

El caso de uso dos (2) suele ser una extensión ***(extend)*** del caso de uso uno (1) cuando realmente se refleja una extensión, de dicho caso de uso uno (1), en el caso de uso dos (2); es decir, el caso de uso dos (2) vendría siendo realmente una extensión del caso de uso uno (1): de su acto o su proceso, de su naturaleza propiamente. Se podría interpretar como el mismo caso de uso uno (1) sólo que con unos *valores agregados* o un *pliego de opciones más avanzado* del mismo; contrario a una inclusión ***(include),*** en una inclusión, el caso de uso dos (2) no es una extensión propia del caso de uso uno (1): es un caso de uso totalmente diferente a lo planteado en el caso de uso uno (1), no trata de un complemento o de una “más detallada especificación” de lo mismo, incluso aunque el caso de uso dos (2) sea una consecuencia del caso de uso uno (1) en algunos planteamientos.

*tip: Por lo general todos los “include” son flechas negras; sin embargo, los “extend” pueden ser flechas negras o blancas.*