Elementos de un diagrama de clases

El diagrama UML de clases está formado por dos elementos: clases, relaciones e interfaces.

Clases

Las clases son el elemento principal del diagrama y representa, como su nombre indica, una clase dentro del paradigma de la orientación a objetos. Este tipo de elementos normalmente se utilizan para representar conceptos o entidades del «negocio». Una clase define un **grupo de objetos** que comparten características, condiciones y significado. La manera más rápida para encontrar clases sobre un enunciado, sobre una idea de negocio o, en general, sobre un tema concreto es buscar los **sustantivos** que aparecen en el mismo. Por poner algún ejemplo, algunas clases podrían ser: Animal, Persona, Mensaje, Expediente… Es un concepto muy amplio y **resulta fundamental identificar de forma efectiva estas clases**, en caso de no hacerlo correctamente se obtendrán una serie de problemas en etapas posteriores, teniendo que volver a hacer el análisis y perdiendo parte o todo el trabajo que se ha hecho hasta ese momento.

Bajando de nivel una clase está compuesta por tres elementos: **nombre de la clase, atributos, funciones. Estos elementos se incluyen en la representación** (o no, dependiendo del nivel de análisis).

Para representar la clase con estos elementos se utiliza una caja que es dividida en tres zonas utilizando para ello lineas horizontales:

- La primera de ellas se utiliza para el **nombre de la clase**. En caso de que la clase sea abstracta se utilizará su nombre en cursiva.
- La segunda, por otra parte, se utiliza para escribir los **atributos** de la clase, uno por línea y utilizando el siguiente formato:

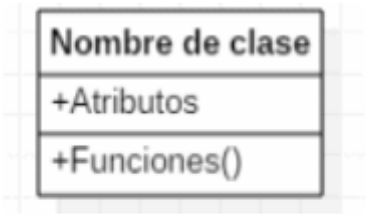
visibilidad nombre_atributo : tipo = valor-inicial { propiedades }

Aunque esta es la forma «oficial» de escribirlas, es común simplificando únicamente poniendo el nombre y el tipo o únicamente el nombre.

- La última de las zonas incluye cada una de las **funciones** que ofrece la clase. De forma parecida a los atributos, sigue el siguiente formato:

visibilidad nombre_funcion { parametros } : tipo-devuelto { propiedades }

De la misma manera que con los atributos, se suele simplificar indicando únicamente el nombre de la función y, en ocasiones, el tipo devuelto.

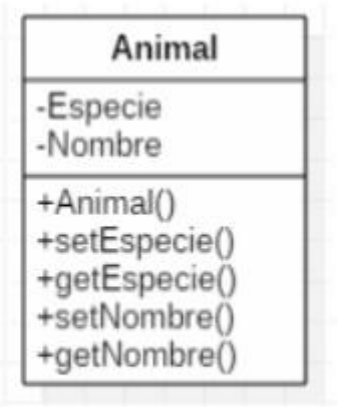


Notación de una clase

Tanto los atributos como las funciones incluyen al principio de su descripción la visibilidad que tendrá. Esta visibilidad se identifica escribiendo un símbolo y podrá ser:

- **(+) Pública.** Representa que se puede acceder al atributo o función desde cualquier lugar de la aplicación.
- **(-) Privada.** Representa que se puede acceder al atributo o función únicamente desde la misma clase.
- **(#) Protegida.** Representa que el atributo o función puede ser accedida únicamente desde la misma clase o desde las clases que hereden de ella (clases derivadas).

- Estos tres tipos de visibilidad son los más comunes. No obstante, pueden incluirse otros en base al lenguaje de programación que se esté usando (no es muy común). Por ejemplo: (/) Derivado o (~) Paquete.
- Un ejemplo de clase podría ser el siguiente:



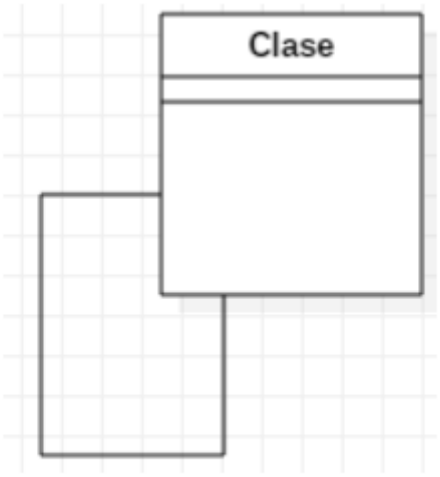
Ejemplo de una clase

En caso de que un atributo o función sea estático, se representa en el diagrama subrayando su nombre. Una característica estática se define como aquella que es

compartida por cada clase y no instanciada para cada uno de los objetos de esa clase. Es un concepto muy común.

Relaciones

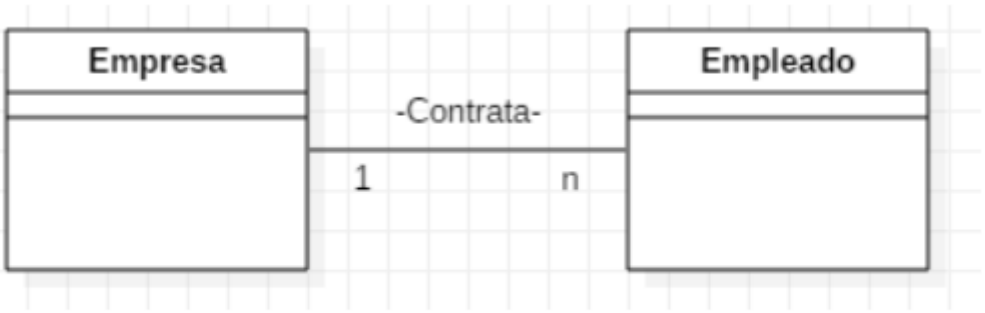
Una relación i**dentifica una dependencia**. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina *dependencia reflexiva*. Las relaciones se representan con una linea que une las clases, esta línea variará dependiendo del tipo de relación



Relación reflexiva

Las relaciones en el diagrama de clases tienen varias propiedades, que dependiendo la profundidad que se quiera dar al diagrama se representarán o no. Estas propiedades son las siguientes:

- **Multiplicidad**. Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango… Se utiliza *n* o * para identificar un número cualquiera.
- **Nombre de la asociación.** En ocasiones se escriba una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como por ejemplo: «Una empresa contrata a n empleados»



Ejemplo de relación Empresa-Empleado

Tipos de relaciones

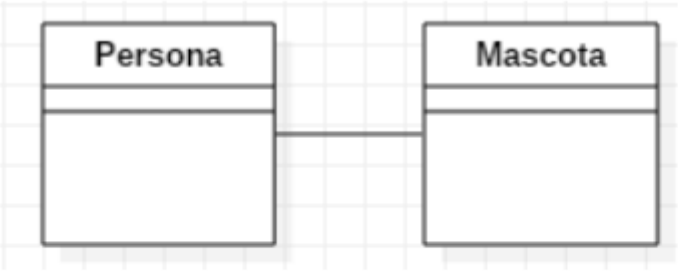
Un diagrama de clases incluye los siguientes tipos de relaciones:

- **Asociación.**
- **Agregación.**
- **Composición.**
- **Dependencia.**
- **Herencia.**

Asociación

Este tipo de relación es el más común y se utiliza para representar dependencia semántica. Se representa con una simple linea continua que une las clases que están incluidas en la asociación.

Un ejemplo de asociación podría ser: «Una mascota pertenece a una persona».



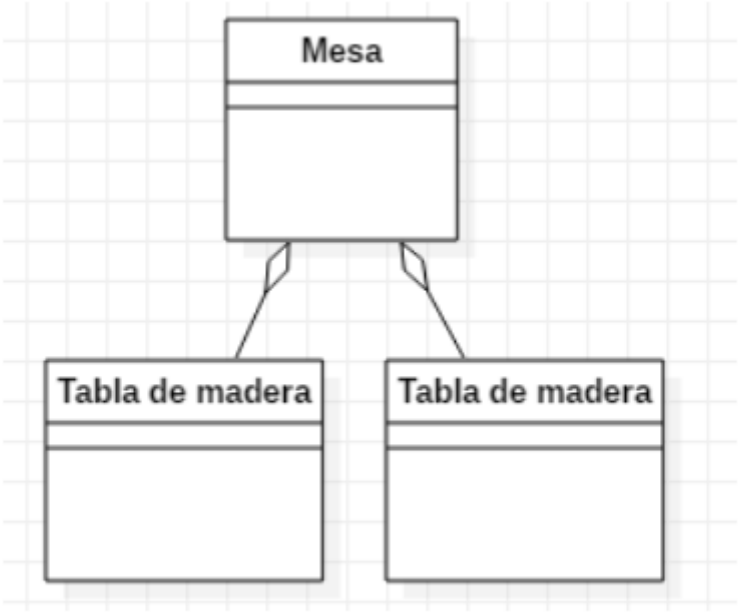
Ejemplo de asociación

Agregación

Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que **un objeto es parte de otro**, pero aun así debe tener **existencia en sí mismo**.

Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Un ejemplo de esta relación podría ser: «Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa». Como ves, el tornillo podría formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.



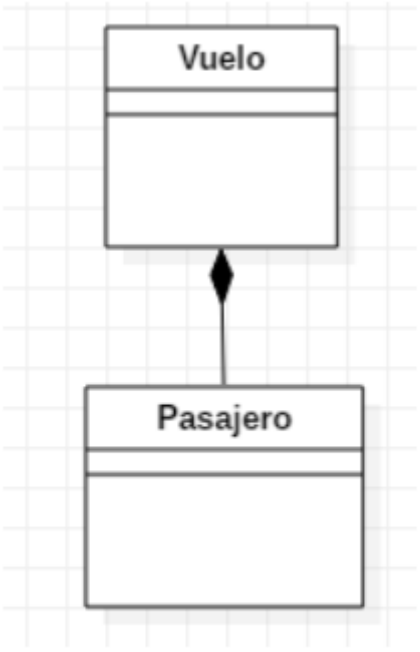
Ejemplo de agregación

Composición

La composición es similar a la agregación, representa una **relación jerárquica entre un objeto y las partes que lo componen**, **pero de una forma más fuerte**. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el primero no existe. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos sino que dependen del elemento que componen. Además, suelen tener los mismos tiempo de vida. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.

Se representa con una linea continua con un rombo relleno en la clase que es compuesta.

Un ejemplo de esta relación sería: «Un vuelo de una compañía aerea está compuesto por pasajeros, que es lo mismo que decir que un pasajero está asignado a un vuelo»



Ejemplo de composición

Diferencia entre agregación y composición

La diferencia entre agregación y composición es semántica, por lo que **a veces no está del todo definida**. Ninguna de las dos tiene análogos en muchos lenguajes de programación (como por ejemplo Java).

Un «agregado» representa **un todo que comprende varias partes**; de esta manera, un Comité es un agregado de sus Miembros. Una reunión es un agregado de una agenda, una sala y los asistentes. En el momento de la implementación, esta relación no es de contención. (Una reunión no contiene una sala). Del mismo modo, las partes del agregado podrían estar haciendo otras cosas en otras partes del programa, por lo que podrían ser referenciadas por varios objetos que nada tienen que ver. En otras palabras, no existe una diferencia de nivel de implementación entre la agregación y una simple relación de «usos». En ambos casos, un objeto tiene referencias a otros objetos. Aunque no existe una diferencia en la implementación, definitivamente vale la pena capturar la relación en el diagrama UML, tanto porque ayuda a comprender mejor el modelo de dominio, como porque puede haber problemas de implementación que pueden pasar desapercibidos. Podría permitir relaciones de acoplamiento más estrictas en una agregación de lo que haría con un simple «uso», por ejemplo.

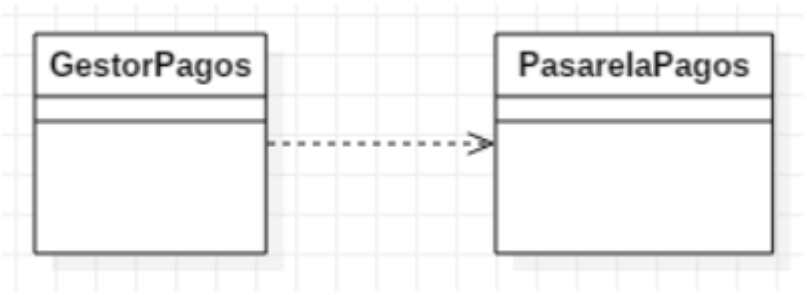
La composición, por otro lado, implica u**n acoplamiento aún más estricto que la agregación**, y definitivamente implica la contención. El requisito básico es que, si una clase de objetos (llamado «contenedor») se compone de otros objetos (llamados «elementos»), entonces los elementos aparecerán y también serán destruidos como un efecto secundario de crear o destruir el contenedor. Sería raro que un elemento no se declare como privado. Un ejemplo podría ser el nombre y la dirección del Cliente. Un cliente sin nombre o dirección no tiene valor. Por la misma razón, cuando se destruye al

cliente, no tiene sentido mantener el nombre y la dirección. (Compare esta situación con la agregación, donde destruir al Comité no debe causar la destrucción de los miembros, ya que pueden ser miembros de otros Comités).

Dependencia

Se utiliza este tipo de relación para **representar que una clase requiere de otra para ofrecer sus funcionalidades**. Es muy sencilla y se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.

Un ejemplo de esta relación podría ser la siguiente:

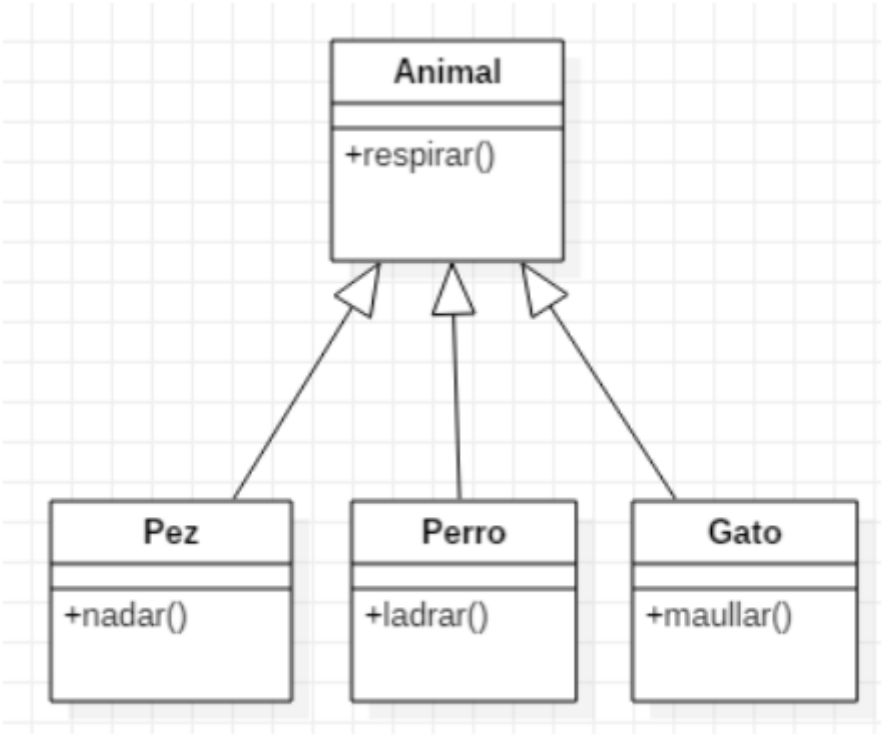


Ejemplo de dependencia

Herencia

Otra relación muy común en el diagrama de clases es la herencia. Este tipo de relaciones permiten que **una clase (clase hija o subclase) reciba los atributos y métodos de otra clase (clase padre o superclase)**. Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma. Se utiliza en relaciones «es un».

Un ejemplo de esta relación podría ser la siguiente: Un pez, un perro y un gato son animales.



Ejemplo de herencia

En este ejemplo, las tres clases (Pez, Perro, Gato) podrán utilizar la función respirar, ya que lo heredan de la clase animal, pero solamente la clase Pez podrá nadar, la clase Perro ladrar y la clase Gato maullar. La clase Animal podría plantearse ser definida abstracta, aunque no es necesario.

This is a modal window.

The media could not be loaded, either because the server or network failed or because the format is not supported.

Diagrama de clases

El **diagrama de clases** es uno de los diagramas incluidos en UML 2.5 clasificado dentro de los diagramas de estructura y, como tal, se utiliza para representar los elementos que componen un sistema de información desde un punto de vista estático.

Es importante destacar que, por esta misma razón, este diagrama no incluye la forma en la que se comportan a lo largo de la ejecución los distintos elementos, esa función puede ser representada a través de un diagrama de comportamiento, como por ejemplo un diagrama de secuencia o un diagrama de casos de uso.

El diagrama de clases es un **diagrama puramente orientado al modelo de programación orientado a objetos**, ya que define las clases que se utilizarán cuando se

pase a la fase de construcción y la manera en que se relacionan las mismas. Se podría equiparar, salvando las distancias, al famoso diagrama de modelo Entidad-Relación (E/R), no recogido en UML, tiene una utilidad similar: la representación de datos y su interacción. Ambos diagramas muestran el modelo lógico de los datos de un sistema.

Elementos de un diagrama de clases

El diagrama UML de clases está formado por dos elementos: clases, relaciones e interfaces.

Clases

Las clases son el elemento principal del diagrama y representa, como su nombre indica, una clase dentro del paradigma de la orientación a objetos. Este tipo de elementos normalmente se utilizan para representar conceptos o entidades del «negocio». Una clase define un **grupo de objetos** que comparten características, condiciones y significado. La manera más rápida para encontrar clases sobre un enunciado, sobre una idea de negocio o, en general, sobre un tema concreto es buscar los **sustantivos** que aparecen en el mismo. Por poner algún ejemplo, algunas clases podrían ser: Animal, Persona, Mensaje, Expediente… Es un concepto muy amplio y **resulta fundamental identificar de forma efectiva estas clases**, en caso de no hacerlo correctamente se obtendrán una serie de problemas en etapas posteriores, teniendo que volver a hacer el análisis y perdiendo parte o todo el trabajo que se ha hecho hasta ese momento.

Bajando de nivel una clase está compuesta por tres elementos: **nombre de la clase, atributos, funciones. Estos elementos se incluyen en la representación** (o no, dependiendo del nivel de análisis).

Para representar la clase con estos elementos se utiliza una caja que es dividida en tres zonas utilizando para ello lineas horizontales:

- La primera de ellas se utiliza para el **nombre de la clase**. En caso de que la clase sea abstracta se utilizará su nombre en cursiva.
- La segunda, por otra parte, se utiliza para escribir los **atributos** de la clase, uno por línea y utilizando el siguiente formato:

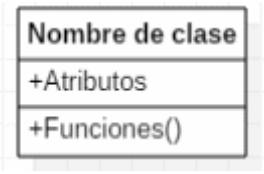
  visibilidad nombre_atributo : tipo = valor-inicial { propiedades }

Aunque esta es la forma «oficial» de escribirlas, es común simplificando únicamente poniendo el nombre y el tipo o únicamente el nombre.

- La última de las zonas incluye cada una de las **funciones** que ofrece la clase. De forma parecida a los atributos, sigue el siguiente formato:

visibilidad nombre_funcion { parametros } : tipo-devuelto { propiedades }

De la misma manera que con los atributos, se suele simplificar indicando únicamente el nombre de la función y, en ocasiones, el tipo devuelto.



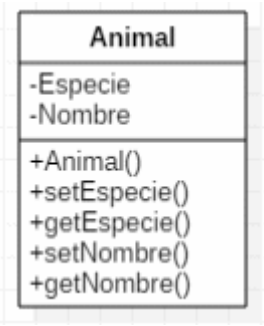
Notación de una clase

Tanto los atributos como las funciones incluyen al principio de su descripción la visibilidad que tendrá. Esta visibilidad se identifica escribiendo un símbolo y podrá ser:

- **(+) Pública.** Representa que se puede acceder al atributo o función desde cualquier lugar de la aplicación.
- **(-) Privada.** Representa que se puede acceder al atributo o función únicamente desde la misma clase.
- **(#) Protegida.** Representa que el atributo o función puede ser accedida únicamente desde la misma clase o desde las clases que hereden de ella (clases derivadas).

Estos tres tipos de visibilidad son los más comunes. No obstante, pueden incluirse otros en base al lenguaje de programación que se esté usando (no es muy común). Por ejemplo: (/) Derivado o (~) Paquete.

Un ejemplo de clase podría ser el siguiente:



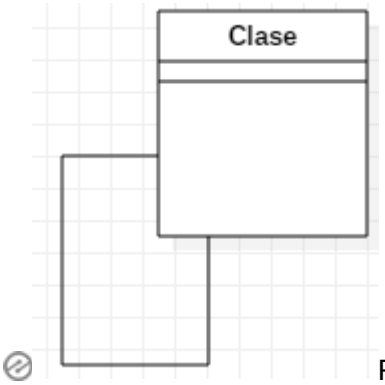
Ejemplo de una clase

En caso de que un atributo o función sea estático, se representa en el diagrama subrayando su nombre. Una característica estática se define como aquella que es compartida por cada clase y no instanciada para cada uno de los objetos de esa clase. Es un concepto muy común.

Relaciones

Una relación i**dentifica una dependencia**. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina *dependencia reflexiva*. Las relaciones se

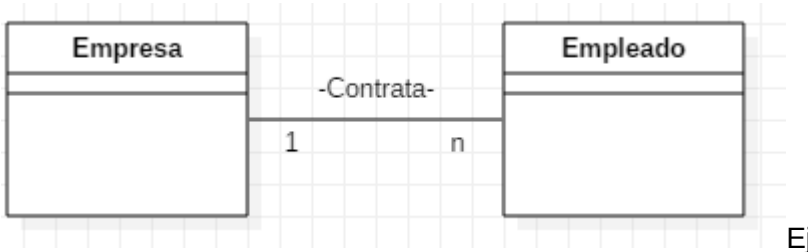
representan con una linea que une las clases, esta línea variará dependiendo del tipo de relación



Relación reflexiva

Las relaciones en el diagrama de clases tienen varias propiedades, que dependiendo la profundidad que se quiera dar al diagrama se representarán o no. Estas propiedades son las siguientes:

- **Multiplicidad**. Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango… Se utiliza *n* o * para identificar un número cualquiera.
- **Nombre de la asociación.** En ocasiones se escriba una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como por ejemplo: «Una empresa contrata a n empleados»



Ejemplo de relación Empresa-Empleado

Tipos de relaciones

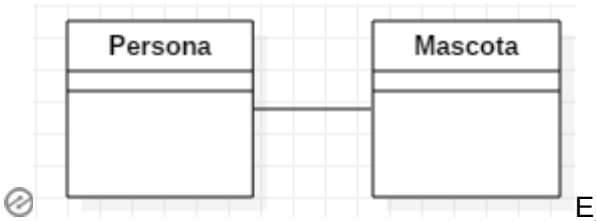
Un diagrama de clases incluye los siguientes tipos de relaciones:

- **Asociación.**
- **Agregación.**
- **Composición.**
- **Dependencia.**
- **Herencia.**

Asociación

Este tipo de relación es el más común y se utiliza para representar dependencia semántica. Se representa con una simple linea continua que une las clases que están incluidas en la asociación.

Un ejemplo de asociación podría ser: «Una mascota pertenece a una persona».



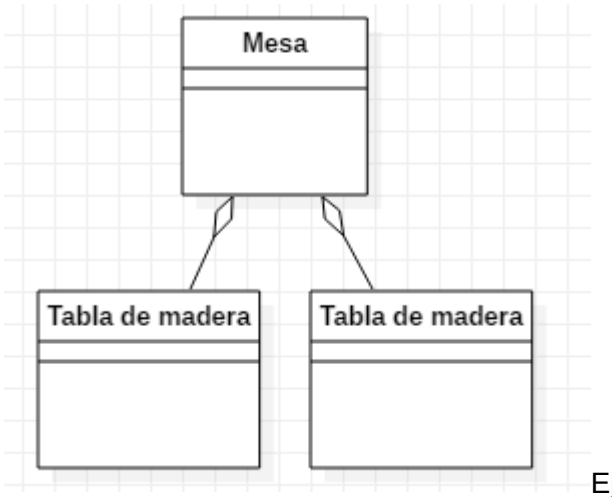
Ejemplo de asociación

Agregación

Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que **un objeto es parte de otro**, pero aun así debe tener **existencia en sí mismo**.

Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Un ejemplo de esta relación podría ser: «Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa». Como ves, el tornillo podría formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.



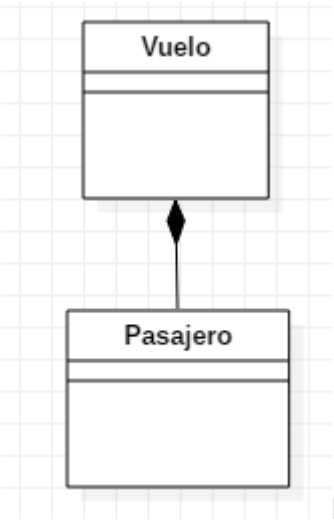
Ejemplo de agregación

Composición

La composición es similar a la agregación, representa una **relación jerárquica entre un objeto y las partes que lo componen**, **pero de una forma más fuerte**. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el primero no existe. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos sino que dependen del elemento que componen. Además, suelen tener los mismos tiempo de vida. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.

Se representa con una linea continua con un rombo relleno en la clase que es compuesta.

Un ejemplo de esta relación sería: «Un vuelo de una compañía aerea está compuesto por pasajeros, que es lo mismo que decir que un pasajero está asignado a un vuelo»



Ejemplo de composición

Diferencia entre agregación y composición

La diferencia entre agregación y composición es semántica, por lo que **a veces no está del todo definida**. Ninguna de las dos tienen análogos en muchos lenguajes de programación (como por ejemplo Java).

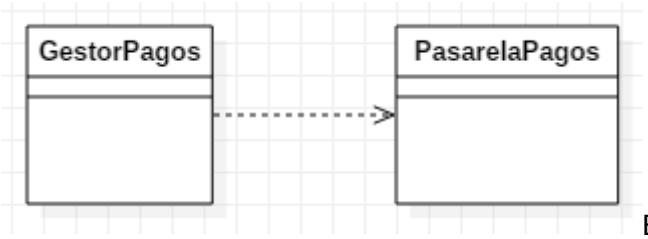
Un «agregado» representa **un todo que comprende varias partes**; de esta manera, un Comité es un agregado de sus Miembros. Una reunión es un agregado de una agenda, una sala y los asistentes. En el momento de la implementación, esta relación no es de contención. (Una reunión no contiene una sala). Del mismo modo, las partes del agregado podrían estar haciendo otras cosas en otras partes del programa, por lo que podrían ser referenciadas por varios objetos que nada tienen que ver. En otras palabras, no existe una diferencia de nivel de implementación entre la agregación y una simple relación de «usos». En ambos casos, un objeto tiene referencias a otros objetos. Aunque no existe una diferencia en la implementación, definitivamente vale la pena capturar la relación en el diagrama UML, tanto porque ayuda a comprender mejor el modelo de dominio, como porque puede haber problemas de implementación que pueden pasar desapercibidos. Podría permitir relaciones de acoplamiento más estrictas en una agregación de lo que haría con un simple «uso», por ejemplo.

La composición, por otro lado, implica u**n acoplamiento aún más estricto que la agregación**, y definitivamente implica la contención. El requisito básico es que, si una clase de objetos (llamado «contenedor») se compone de otros objetos (llamados «elementos»), entonces los elementos aparecerán y también serán destruidos como un efecto secundario de crear o destruir el contenedor. Sería raro que un elemento no se declare como privado. Un ejemplo podría ser el nombre y la dirección del Cliente. Un cliente sin nombre o dirección no tiene valor. Por la misma razón, cuando se destruye al cliente, no tiene sentido mantener el nombre y la dirección. (Compare esta situación con la agregación, donde destruir al Comité no debe causar la destrucción de los miembros, ya que pueden ser miembros de otros Comités).

Dependencia

Se utiliza este tipo de relación para **representar que una clase requiere de otra para ofrecer sus funcionalidades**. Es muy sencilla y se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.

Un ejemplo de esta relación podría ser la siguiente:

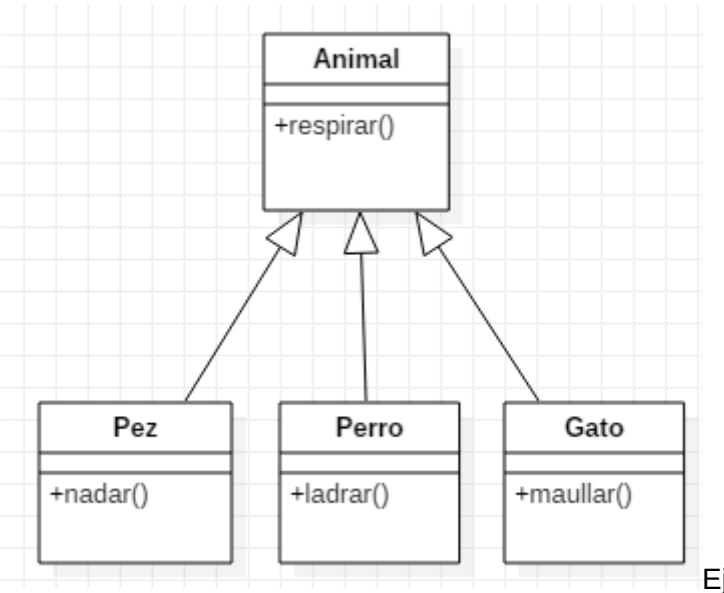


Ejemplo de dependencia

Herencia

Otra relación muy común en el diagrama de clases es la herencia. Este tipo de relaciones permiten que **una clase (clase hija o subclase) reciba los atributos y métodos de otra clase (clase padre o superclase)**. Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma. Se utiliza en relaciones «es un».

Un ejemplo de esta relación podría ser la siguiente: Un pez, un perro y un gato son animales.



Ejemplo de herencia

En este ejemplo, las tres clases (Pez, Perro, Gato) podrán utilizar la función respirar, ya que lo heredan de la clase animal, pero solamente la clase Pez podrá nadar, la clase Perro ladrar y la clase Gato maullar. La clase Animal podría plantearse ser definida abstracta, aunque no es necesario.

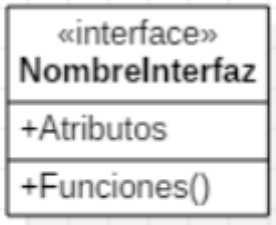
Interfaces

Una interfaz es una entidad que declara una **serie de atributos, funciones y obligaciones.** Es una especie de contrato donde toda instancia asociada a una interfaz debe de implementar los servicios que indica aquella interfaz.

Dado que únicamente son declaraciones **no pueden ser instanciadas**.

Las interfaces se asocian a clases. Una asociación entre una clase y una interfaz representa que esa clase cumple con el contrato que indica la interfaz, es decir, incluye aquellas funciones y atributos que indica la interfaz.

Su representación es similar a las clases, pero indicando arriba la palabra <<interface>>.



Notación de interfaz

Cómo dibujar un diagrama de clases

Los diagramas de clase van de la mano con el diseño orientado a objetos. Por lo tanto, saber lo básico de este tipo de diseño es una parte clave para poder dibujar diagramas de clase eficaces.

Este tipo de diagramas son solicitados cuando se está describiendo la vista estática del sistema o sus funcionalidades. Unos pequeños pasos que puedes utilizar de guía para construir estos diagramas son los siguientes:

- **Identifica** los nombres de las clase
  El primer paso es identificar los objetos primarios del sistema. Las clases suelen corresponder a sustantivos dentro del dominio del problema.
- **Distingue** las relaciones
  El siguiente paso es determinar cómo cada una de las clases u objetos están relacionados entre sí. Busca los puntos en común y las abstracciones entre ellos; esto te ayudará a agruparlos al dibujar el diagrama de clase.
- **Crea** la estructura
  Primero, agrega los nombres de clase y vincúlalos con los conectores apropiados, prestando especial atención a la cardinalidad o las herencias. Deja

los atributos y funciones para más tarde, una vez que esté la estructura del diagrama resuelta.

Buenas prácticas en la construcción del diagrama de clases

Te recomendamos seguir estas indicaciones o consejos, que, aunque no son obligatorios, harán que tus diagramas de clases sean de mayor utilidad:

- Los diagramas de clase **pueden tender a volverse incoherentes** a medida que se expanden y crecen. Es mejor evitar la creación de diagramas grandes y **dividirlos** en otros más pequeños que se puedan vincular entre sí más adelante.
- Usando la notación de clase simple, puedes crear rápidamente **una visión general de alto nivel** de su sistema. Se puede crear un diagrama detallado por separado según sea necesario, e incluso vincularlo al primero para una referencia fácil.
- Cuantas más líneas se superpongan en sus diagramas de clase, más abarrotado se vuelve y, por tanto, más se complica utilizarlo. El lector se confundirá tratando de encontrar el camino. Asegúrate de que **no haya dos líneas cruzadas** entre sí, a no ser que no haya más remedio.
- Usa **colores** para agrupar módulos comunes. Diferentes colores en diferentes clases ayudan al lector a diferenciar entre los diversos grupos.