# Diagrama de Clases

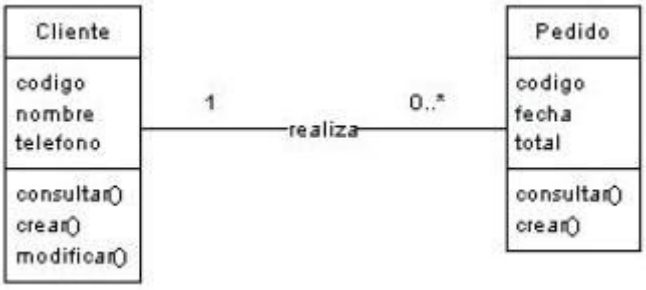
- **Clases**: Cada elemento de nuestro sistema de información del cual vayamos a guardar información en la BBDD será representado mediante una clase.
    - Atributos: definiremos los atributos que necesitamos guardar de ese elemento *(clase Cliente: Nif, nombre, apellidos, etc)*
    - Métodos: contendrá los métodos CRUD necesarios que actuarán sobre los atributos de esta clase, además de métodos adicionales que sean necesarios. *Estos métodos se utilizarán en los diagramas de interacción, así que deben estar los necesarios para poder abordar los requisitos funcionales.*
    - Tanto los atributos como los métodos pueden tener distintos tipos de visibilidad. Generalmente los atributos tienen visibilidad privada y los métodos tendrán visibilidad pública.
        - \+ Visibilidad pública: Visible por todos los objetos
        - \- Visibilidad privada: Visible sólo por el objeto
        - # Visibilidad protegida: Visible sólo por el objeto y sus descendientes

**Relaciones**

Definición: Cuando las clases se conectan entre sí de forma conceptual, esta conexión se conoce como relación.

Cardinalidad: las relaciones pueden tener diferentes tipos de cardinalidad:

- * (equivalente a 0..*)
- n..*
- n..m
- n

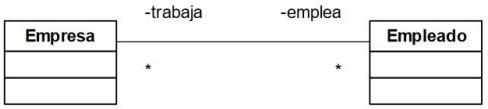



En este ejemplo un cliente puede hacer 0 o muchos pedidos y es realizado por un único cliente

Hay diferentes tipos de relaciones:

- **Relaciones de asociación:**
    - Definición: Diremos que dos (o más) clases tienen una relación de asociación cuando una de ellas tenga que utilizar alguno de los servicios (es decir, acceder a alguna de las atributos o métodos) de las otras.

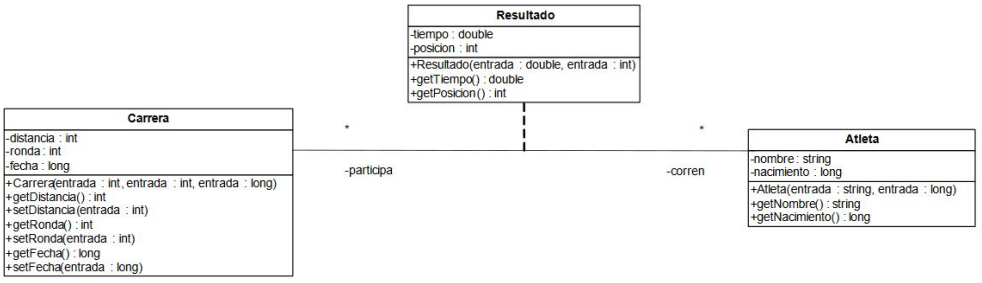
- IMPORTANTE: Las relaciones de *asociación* son más débiles que las relaciones de *agregación/composición*, ya que en las relaciones de asociación únicamente se necesita que los objetos interactúen entre sí.
- Multiplicidad: Se indicará cómo se relacionan las clases mediante la cardinalidad en la relación.
- EJEMPLO:




*Los objetos deben EXISTIR fuera de la relación (a diferencia de las relaciones de composición).*

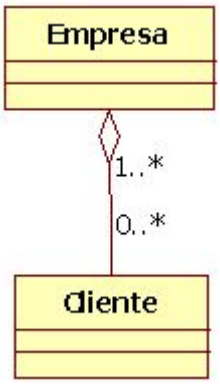
- **Clase de Asociación:** También podemos encontrar relaciones de asociación en las que no sea suficiente con representar la relación entre dos clases, sino que además debamos contemplar la aparición de propiedades adicionales vinculadas a la relación de asociación.

- EJEMPLO:



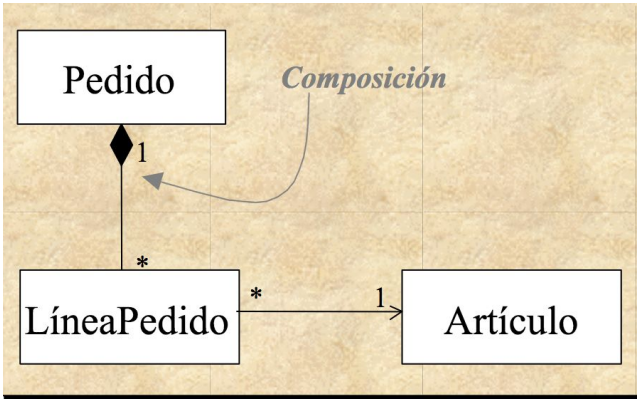

En este caso, nos interesa conocer el resultado de los atletas en diversas carreras. Para ello definimos lo que se conoce como “Clase de Asociación”, que nos permite vincular dos objetos particulares de dos clases (en este caso, de las clases ”Atleta” y “Carrera”) y además añadir cierta información relevante al sistema que no tiene por qué estar vinculada a ninguna de ellas (en este caso, el resultado de un atleta en una carrera).

- **Relaciones de agregación:**
  - Definición: Los componentes y la clase que constituyen son una asociación que conforma un todo. Si el objeto base desaparece NO desaparecen los objetos incluidos.
  - ¿Cuando poner asociación o agregación? Cuando conceptualmente un objeto es parte de otro ponemos agregación.
  - EJEMPLO: Pedido-Productos, Categoría-Productos



Una empresa agrupa varios clientes

- **Relaciones de composición:**
  - Definición: Cada objeto dentro de una composición puede pertenecer SOLAMENTE a un objeto base. Si el objeto base desaparece SI desaparecen los objetos incluidos.
  - IMPORTANTE: En una composición el objeto compuesto y los componentes se crean en el momento. Es decir, los componentes no existen en el sistema hasta que el objeto que los contiene se haya creado. Por ejemplo, cuando creo un Pedido, cada línea de pedido necesaria se crea en el momento, no se puede seleccionar una línea de pedido que ya exista en el sistema.
  - Ejemplos: Pedido-LineaPedido, Libro-capítulos, SitioWeb-PaginasWeb
  - EJEMPLO:



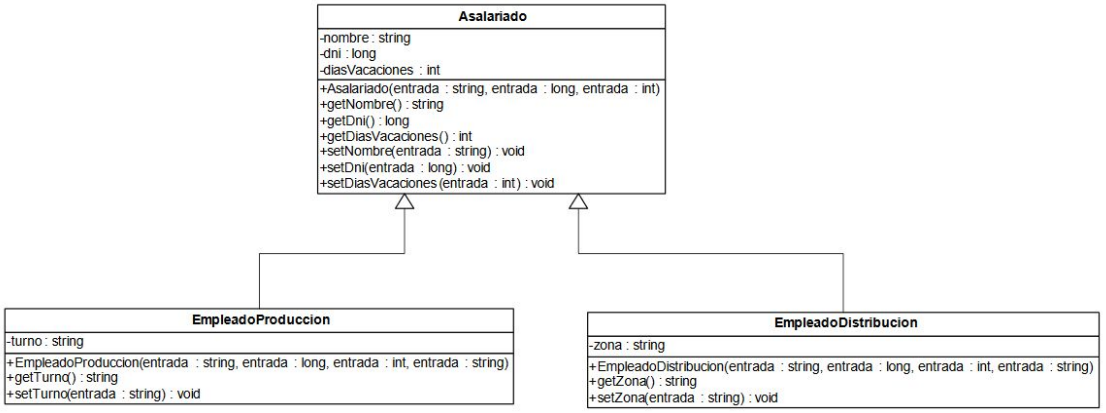

  - **No** son composiciones: Cliente - Factura, Cliente-Pedido,

Diferencias entre agregación y composición:

| | **Agregación** | **Composición** |
|---|---|---|
| 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 ó 1 |
| Representación | Rombo transparente | Rombo negro |

- **Generalización (Especialización/Herencia)**
  - Definición: Indica que una clase (clase derivada) hereda los métodos y atributos especificados por una clase (clase base), por lo cual una clase derivada además de tener sus propios métodos y atributos, podrá acceder a las características y atributos visibles de su clase base (public y protected).

  - Ejemplo

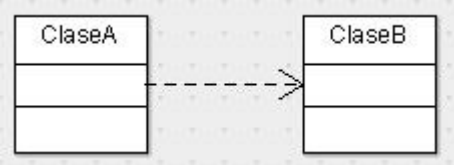



Definiremos una clase “Asalariado” que nos permita representar a todos los trabajadores de una empresa. A partir de esa clase, definiremos dos subclases “EmpleadoProduccion” y “EmpleadoDistribucion” que representan a los trabajadores de dicha empresa que trabajan en las divisiones de producción y distribución respectivamente. Los trabajadores de producción trabajan por turnos (“Mañana”, “Tarde”, “Noche”). Los de distribución tienen asignada una zona (o región) donde deben distribuir los productos, pero no trabajan por turnos.
En primer lugar, podemos verificar que las clases están vinculadas por una relación de herencia. En particular, un objeto de la clase “EmpleadoProduccion” “es - un” “Asalariado”, del mismo modo que un objeto de la clase “EmpleadoDistribucion” “es - un” “Asalariado”.

Todos los atributos que aparecen en la clase base (“Asalariado” en nuestro caso) no deben aparecer en las clases derivadas. De igual manera, los métodos que

aparezcan en la clase base y cuyo comportamiento no varíe en las clases derivadas tampoco aparecerán en las mismas. En las clases derivadas sólo debemos incluir los atributos y métodos que no aparecían en la clase base (y aquéllos que redefinamos)

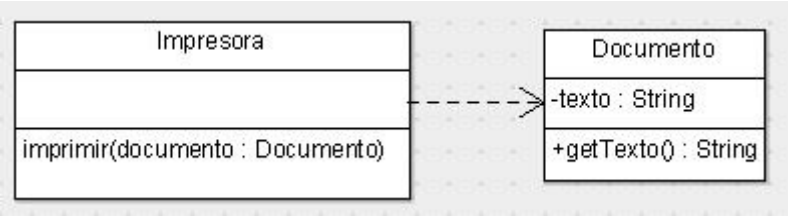
- **Dependencias**
    - Definición: Es una relación de uso entre dos clases (una usa a la otra). Esta relación es la más básica entre clases y comparada con los demás tipos de relación, la mas débil. Se representa con una flecha discontinua que parte desde una clase y apunta a otra. El sentido de la flecha nos indica quien usa a quien.




- **La ClaseA usa a la ClaseB.**
- **La ClaseA depende de la ClaseB.**
- **Dada la dependencia, todo cambio en la ClaseB podrá afectar a la ClaseA.**
- **La ClaseA conoce la existencia de la ClaseB pero la ClaseB desconoce que existe la ClaseA.**

En la practica este tipo de relación se interpreta como que la ClaseA hace uso de la ClaseB ya sea instanciandola directamente, o bien, recibiéndola como parámetro de entrada en uno de sus métodos.

- Ejemplo




1. Tenemos una clase Impresora..
2. Tenemos una clase Documento con un atributo texto.
3. La clase Impresora se encarga de imprimir los Documentos.

Fuentes:
http://www.unirioja.es/cu/jearansa/0910/archivos/EIPR_Tema02.pdf
https://repositorio.grial.eu/bitstream/grial/353/1/DClase.pdf