**Diagramas UML**

Cuando para el desarrollo de una aplicación se decide utilizar un lenguaje de programación orientado a objetos, para representar de forma visual tanto las clases que serán utilizadas como las relaciones existentes entre ellas se usa el **lenguaje UML** (Unified Modeling Language, o Lenguaje de Modelado Unificado en español), que define una serie de reglas para crear **diagramas UML**.

Es recomendable realizar previamente un análisis del problema a resolver, así como de los requisitos y pasos que debe cumplir la aplicación para que una vez finalizada cumpla con su objetivo sin que se produzcan errores.

***Con UML se pueden crear los siguientes tipos de diagramas:***

* **Estáticos**
  + **De clases**: modelan la estructura estática de las clases.
  + **De objetos**: para modelar la estructura estática de los objetos.
  + **De componentes**: para modelar los componentes.
  + **De despliegue**: modelan la distribución del sistema.
* **Dinámicos**
  + **De casos de uso**: modelan los procesos de negocio.
  + **De secuencia**: modelan el traspaso de mensajes entre objetos.
  + **De colaboración**: representan las interacciones entre los objetos.
  + **De estados**: representan el comportamiento de casos de uso, objetos y operaciones.
  + **De actividades**: modelan el comportamiento de los casos de uso, objetos u operaciones.

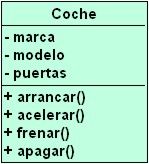
Por último, para crear los diagramas puedes utilizar los programas gratuitos ArgoUML, StarUML, OpenAmeos o Umbrello, UML Modeller por ejemplo. Tambien hay sitios web donde poder hacerlos: LucidChart, Dia, etc.

**Representacion de clases**

Pongamos por ejemplo una Clase 'Coche' que tiene como atributos o propiedades 'marca', 'modelo', 'bastidor'... y como métodos ‘arrancar ()', ‘acelerar ()'...

Las clases se representan escribiendo como encabezado en un rectángulo el nombre de la misma, seguido por los nombres de los atributos y a continuación los de sus acciones o métodos.

El nombre de la clase comenzará con letra mayúscula. Cuando se trate de palabras compuestas se unen ambas quedando en mayúscula la primera letra de cada una de ellas:



Los **nombres de atributos y métodos** comienzan con letra minúscula. Cuando se trate de palabras compuestas se unen quedando sólo en mayúscula la primera letra de la segunda palabra.

El signo '-' delante de sus nombres indica que su visibilidad es privada (private) y el '+' que es pública (public). Cuando es protegido (protected) se utiliza la almohadilla '#'. Si estamos utilizando lenguajes como Java con '~' indicamos que la visibilidad es sólo a nivel del **paquete** en el que se encuentra la clase.

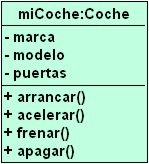
Por otro lado, para representar una **Clase abstracta** (abstract) escribiremos **<<abstract>>** sobre su nombre en cursiva:



Asimismo, se escriben en cursiva los nombres de los métodos que no se encuentren implementados en la misma, debiendo serlo en las clases que hereden de ella (puede tener algunos ya implementados pero otros que no lo estén).

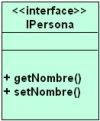
**Representar objetos**

Para representar una **Instancia u objeto** de una Clase escribiremos en el encabezado el nombre del mismo (comenzando con minúscula) seguido por dos puntos y a continuación el nombre de la clase:



**Interfaces**

Las **Interfaces** se representan escribiendo el nombre en cursiva y opcionalmente sobre él **<<interface>>**, constando también los nombres de los métodos que deberán ser definidos en la clase que la implemente (en la interface no se implementa ninguno):



También se puede representar mediante un círculo y escribiendo su nombre junto a él:

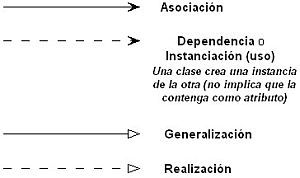
Interface

**Relación entre elementos**

Las relaciones entre los elementos utilizados en un diagrama de clases se definen mediante líneas o flechas. Dichas relaciones pueden ser de:

* **Asociación**
* **Dependencia**
* **Generalización**
* **Realización**

Cada tipo de relación se representa en un diagrama con un tipo de flecha:

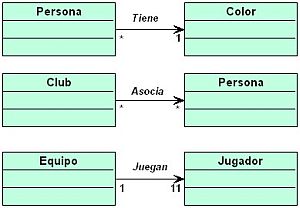


En la **Asociación** se especifica que dos clases u objetos están relacionados pero siendo un elemento independiente del otro.

Mediante la **multiplicidad o cardinalidad** se indica cuántas instancias de una clase pueden surgir de la relación, y se indican mediante:

* 1 --> sólo uno
* 0..1 --> cero o uno
* n --> mediante un número entero se indica cuántas relaciones pueden haber.
* n..m --> varios a varios
* \* --> cero o más
* 0..\* --> cero o más (lo mismo que el anterior)
* 1..\* --> uno o más

*Observemos los siguientes ejemplos:*

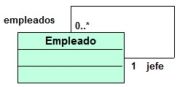


En el primer ejemplo de la imagen se define que una instancia de 'Persona' puede estar relacionada sólo con una de 'Color', que a su vez puede estar relacionada con varias instancias de 'Persona'.

En el segundo, una instancia de 'Club' puede estar relacionada con varias de 'Persona', y cada una de ellas puede estar relacionada con varias de 'Club'.

En el tercero, una instancia de 'Equipo' puede estar relacionada con 11 instancias de 'Jugador', y cada una de ellas a su vez con sólo una de 'Equipo'.

Por otro lado, si la asociación es entre objetos de la misma clase nos encontramos ante una **asociación reflexiva**. En la imagen se muestra que un empleado jefe está relacionado con ninguno o varios empleados, y que un empleado tiene sólo un jefe:



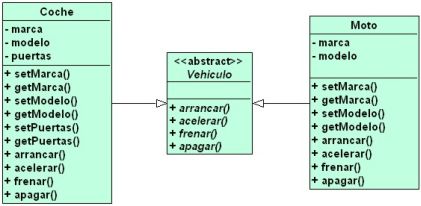
En la **Dependencia** entre dos clases uno de los elementos es dependiente ('Coche' según la imagen de ejemplo) del otro, viéndose afectado por los cambios que se produzcan en él y no pudiendo funcionar correctamente si le falta:

Dependencia

Ello no implica que el elemento dependiente deba obligatoriamente contener al otro como uno de sus atributos (puede que lo necesite como parámetro o que lo utilice en un método, por ejemplo).

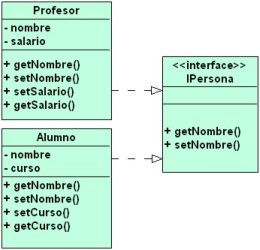
En lugar de la **Dependencia** es recomendable definir la relación como **Asociación**, a menos que sea importante representarla como tal debido a que un cambio en un elemento del que depende(n) otro(s) pueda afectar al funcionamiento del elemento dependiente.

En cuanto a la **Generalización**, indica una relación de **herencia** entre dos clases u objetos (el elemento hijo comparte la misma estructura que el elemento padre):



Mediante la **Realización** uno de los elementos define características que deberá desarrollar el que lo implemente. En los diagramas de clases se utilizan para representar las relaciones con **interfaces**.

En la siguiente imagen se representa cómo dos clases implementan la interfaz 'IPersona':



La relación se puede representar también la clase con la interface representada mediante un círculo. De este modo también indica que la clase 'Profesor' implementa la interfaz 'IPersona':



**Otros tipos de relaciones**

Aparte de las comentadas en el punto anterior existen dos variantes de las relaciones de Asociación y que dadas sus definiciones suelen resultar algo ambiguas: **Agregación y Composición.**

En la **Agregación** el conjunto de elementos relacionados forman un "TODO", aunque los elementos relacionados podrían existir y "funcionar" independientemente de él. Se representan mediante una línea con un rombo sin relleno en un extremo.

En este ejemplo, si los departamentos de una institución educativa cerrasen los profesores continuarían existiendo como tales (el departamento al estar conectado por el rombo representa el "TODO"):

Agregación

Por otro lado la **Composición** es básicamente lo mismo, pero los elementos relacionados no podrían existir y "funcionar" independientemente si no formasen parte de dicho "TODO". Se representan mediante una línea con un rombo con relleno en un extremo.

Según la imagen de ejemplo, el cierre de la universidad provocaría que desapareciesen también los departamentos (la universidad al estar conectada con el rombo representa el "TODO"):

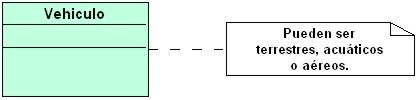
Composición

**Notas en los Diagramas**

Si deseamos agregar algún texto con información a nuestros diagramas de clases y objetos, insertaremos el mismo en un rectángulo con una de sus esquinas doblada:

Nota

Las notas pueden estar en cualquier parte del diagrama sin mantener relación con ningún elemento en particular, aunque se utilizar una línea discontinua para especificar a qué elemento hacen referencia:



Draw.io

LucidChart

Star UML

UMLet