



**CICLO 2**

[FORMACIÓN POR CICLOS]


# Diagramas de **CLASES UML**



**Ingeni@**  
Soluciones TIC



**UNIVERSIDAD  
DE ANTIOQUIA**  
Facultad de Ingeniería



La programación orientada a objetos (POO), como paradigma de programación, se apoya en el diseño orientado a objetos. Cuando hablamos de diseño en este contexto, no nos referimos necesariamente al diseño artístico o visual de una aplicación, sino al modelado y representación de su estructura. Específicamente, el diseño orientado a objetos consiste en un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para bosquejar modelos lógicos y físicos, así como estáticos y dinámicos del sistema que se está diseñando. En otras palabras, se trata de descomponer en objetos un sistema y/o su dominio, representándolos mediante una notación apropiada.

Para el modelado de sistemas de información se usa principalmente el estándar UML, que desde 2017 se encuentra en su versión 2.5.1. Dicho estándar propone y dicta la especificación de varios diagramas usados para el modelado de sistemas de información. Los diagramas UML pueden ser estáticos o dinámicos.

Por un lado, los diagramas estáticos se ocupan de representar la estructura de objetos, clases y componentes que comprenden un dominio o una aplicación. Por el otro, los diagramas dinámicos representan cómo interactúan clases, objetos, componentes y actores en el tiempo de ejecución del sistema o aplicación. Uno de los diagramas UML estáticos más usados, y piedra angular del diseño orientado a objetos, es el diagrama de clases UML.

Un **diagrama de clases** se puede utilizar, principalmente, para dos cosas:

1. Modelar un dominio, es decir, modelar los conceptos, propiedades y relaciones que comprenden el área de conocimiento o de práctica en la cual se enmarca la aplicación de *software* en desarrollo.
2. Representar y visualizar la estructura de las clases que componen un sistema y las relaciones entre ellas.

En el segundo caso podríamos tener, además, dos propósitos:

1. Representar la estructura de un programa ya existente.
2. Representar la estructura de un programa que apenas vamos a desarrollar.



Y su sintaxis, como vimos arriba, se rige por el estándar UML. Dicho estándar determina cómo se representa gráficamente la mayor parte de los conceptos de la programación orientada a objetos, e incluso hay una equivalencia fuerte entre lo que se modela usando UML y lo que se programa usando lenguajes como Java.


A pesar de lo anterior, la notación UML para diagramas de clases tiene muchas variantes. Si hacemos el ejercicio de buscar en internet cómo se hace un diagrama de clases UML, encontraremos tantas notaciones diferentes como resultados. A continuación, haremos un repaso de los elementos más comunes de la notación más común, pudiendo luego construir el diagrama de clases de un dominio o de una aplicación de la misma manera.

En UML, el elemento base es la **clase**. Una clase se representa como una caja compuesta a su vez por tres cajones. En el primer cajón, el de más arriba, se escribe el nombre de la clase, en negrita; en el segundo cajón, se listan los atributos; y en el tercero, se listan los métodos de la clase.

A continuación, tenemos las clases *Aeropuerto* y *Auto*.

Aeropuerto	Auto
- ciudad: String	+ placa: String + modelo: String
+ setCiudad(String): void + getCiudad(): String	+ bloquear()





En el ejemplo de arriba, ambas clases tienen atributos y métodos. El cómo se representan los métodos y atributos varía entre diagramas, y depende en gran medida de la herramienta usada para el modelado, de quien realiza el modelo, e incluso del lenguaje de programación en el que se piensa al realizar el modelo. En el ejemplo de arriba, los **atributos** se representan así:

- En primer lugar, se pone un símbolo que depende de si el atributo es de acceso público (+), protegido (#) o privado (-).
- Luego, se pone el nombre del atributo, seguido de dos puntos (:).
- Y, finalmente, se pone el tipo de dato del atributo, que para todos los casos de arriba serán atributos de tipo String.

Por su parte, los **métodos** se representan de forma muy similar, así:

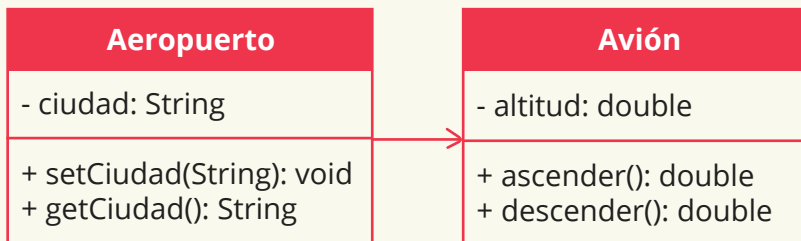
- En primer lugar, se pone un símbolo que depende de si el método es de acceso público (+), protegido (#) o privado (-).
- Luego, se pone el nombre del método.
- Después, se pueden especificar entre paréntesis los parámetros del método, seguido de dos puntos (:).
- Y, finalmente, se pone el tipo de dato que retorna el método. Los tipos de datos que se podrán poner en este punto dependerán enteramente del lenguaje de programación, por lo cual, por ejemplo, en el diagrama de arriba tenemos un método tipo void y otro tipo String (tipos propios del lenguaje Java).

Además de las clases con sus atributos y métodos, en los diagramas de clases es posible representar cuatro tipos de relaciones entre clases: asociación, agregación, composición y herencia.

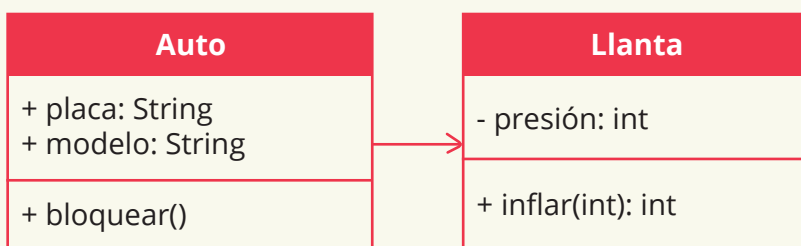
La relación de **asociación** se representa usando una flecha convencional, como la que se muestra abajo, y sirve para representar una relación débil entre dos clases, sin un significado especial. En este caso, existe una asociación entre las clases *Avión* y *Aeropuerto*.



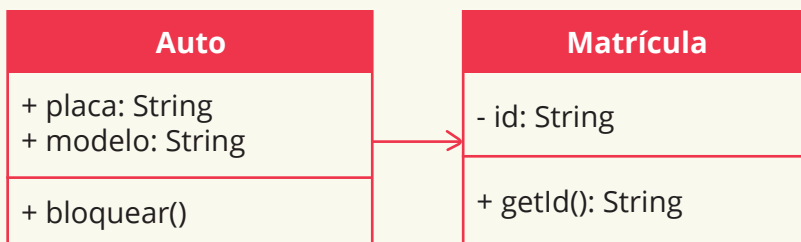




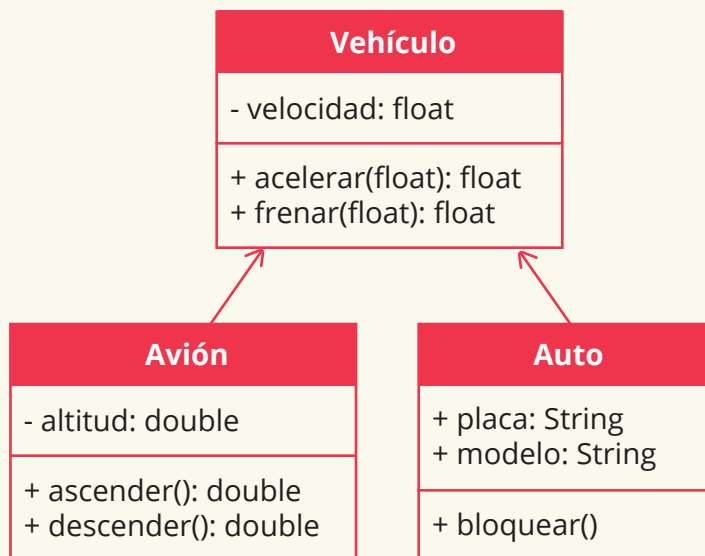
La relación de **agregación**, tal como se muestra abajo, empieza con un rombo vacío (blanco) y finaliza en una flecha. Este tipo de relación representa una relación más fuerte, pero en la cual el ciclo de vida de un objeto de la clase donde termina la relación no depende del ciclo de vida de un objeto de la clase donde inicia. En este caso, hay una relación fuerte entre *Auto* y *Llanta*, pero el ciclo de vida de un objeto de la clase *Llanta* no requerirá que exista un objeto de la clase *Auto*.



La relación de **composición** es similar a la de agregación, iniciando en un rombo lleno (negro). En este caso, la relación entre ambas clases es aún más fuerte, y el ciclo de vida de un objeto de la clase donde termina la relación siempre dependerá del ciclo de vida de un objeto de la clase donde inicia esta. Por ejemplo, en este caso, un objeto de la clase *Matrícula* exigirá que exista un objeto de la clase *Auto*.



Por último, tenemos la relación de **herencia**. Esta siempre se representa usando una flecha como las que se muestran a continuación, y siempre representan la relación entre una o varias clases hijas y su clase padre. Recordemos que, en una relación de herencia, las clases hijas heredan los métodos y atributos de su clase padre, pudiendo agregar a su vez sus propios métodos y atributos.



Finalmente, a continuación, tenemos un diagrama de clases UML simple, pero que cuenta con los elementos más comunes en diagramas de este tipo. Observe con detenimiento las clases, métodos, atributos y relaciones representadas en el diagrama, y se hará una idea del dominio que se quiere representar.

