Entornos de desarrollo

ÍNDICE

1. Introducción

El **diagrama de clases** puede considerarse el más importante de todos los existentes en UML, encuadrado dentro de los diagramas estructurales, representa los elementos estáticos del sistema, sus atributos y comportamientos, y como se relacionan entre ellos. A partir de éste se obtendrán las clases que formarán después el programa informático.

En un diagrama de clases podemos encontrar los siguientes elementos:

- **Clases**: agrupan conjuntos de *objetos* con características comunes, que llamaremos *atributos*, y su comportamiento, que serán *métodos*. Los atributos y métodos tendrán una visibilidad que determinará quién puede acceder al atributo o método. Por ejemplo, una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará

- **Relaciones**: en el diagrama se representan relaciones reales entre los elementos del sistema a los que hacen referencia las clases. Pueden ser de *asociación, agregación, composición y generalización*. Por ejemplo, si tenemos las clases persona y coche, se puede establecer la relación conduce entre ambas. O una clase alumno puede tener una relación de generalización respecto a la clase persona.

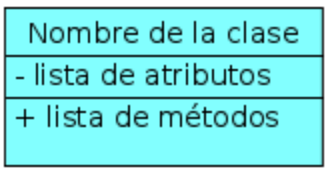
- **Notas**: se representan como un cuadro donde podemos escribir comentarios que ayuden al entendimiento del diagrama.

- **Elementos de agrupación**: se utilizan cuando hay que modelar un sistema grande, entonces las clases y sus relaciones se agrupan en paquetes, que a su vez se relacionan entre sí.

2. Creación de Clases

Una clase se representa en el diagrama como un rectángulo divido en **tres** partes:

- Arriba aparece el nombre de la clase,
- Medio los atributos
- Abajo los métodos



3. Atributos

Forman la parte estática de la clase. Son un conjunto de **variables** para las que es preciso definir:

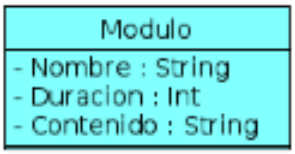
- Su **nombre**.

- Su **tipo**, puede ser un tipo simple, que coincidirá con el tipo de dato que se seleccione en el lenguaje de programación final a usar, o compuesto, pudiendo incluir otra clase.

Además, se pueden indicar otros datos como un **valor inicial** o su **visibilidad**.
La **visibilidad** de un atributo se puede definir como:

- **Público (+)**: se pueden acceder desde cualquier clase y cualquier parte del programa.
- **Privado (-)**: sólo se pueden acceder desde operaciones de la clase.
- **Protegido (#)**: sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
- **Paquete (~)**: se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación es Java.

Ejemplo: crear una clase de nombre "Módulo" y que tenga atributos nombre, duración y contenidos con visibilidad privado:



Nombre, de tipo String.
Duracion de tipo Int.
Contenidos de tipo String

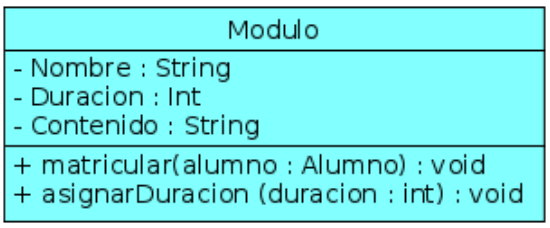
4. Métodos

Representan la **funcionalidad** de la clase, es decir, qué puede hacer. Para definir un método hay que indicar como mínimo su nombre, parámetros, el tipo que devuelve y su visibilidad (similar a la de los atributos). También se debe incluir una descripción del método que aparecerá en la documentación que se genere del proyecto.

Existe dos métodos particulares:

- El **constructor** de la clase, que tiene como característica:
  - No devuelve ningún valor.
  - Tiene el mismo nombre de la clase
  - Ejecutar las acciones necesarias cuando se instancia un nuevo objeto, es decir, reservar memoria e inicializar valores en caso de que se desee. (new)
  - Puede haber varios constructores según las necesidades. (Incluso puede existir uno vacío)
  -
- El **destructor** de la clase. Cuando no se vaya a utilizar más el objeto, se podrá utilizar un método destructor que libere los recursos del sistema que tenía asignados. La destrucción del objeto es tratada de formas diferentes en función del lenguaje de programación que se esté utilizando. Concretamente en Java no se existen los destructores porque existe el recolector de basura que se encarga de liberar memoria cuando una instancia deja de ser referenciada.
-

Ejemplo: añadir a la clase creada anteriormente los métodos matricular y asignarDuracion con visibilidad pública:

	matricular(alumno : Alumno) : void asignarDuración(duracion : int) : void

5. Relaciones entre clases

Una **relación** es una conexión entre dos clases que incluimos en el diagrama.

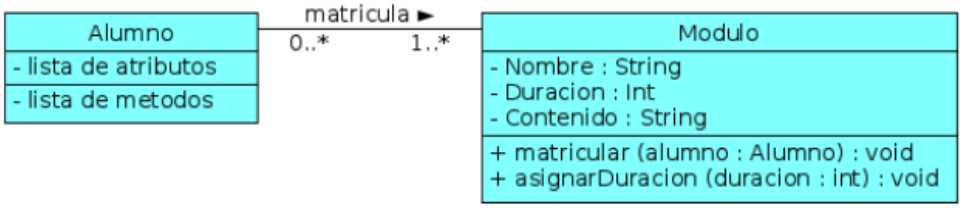
Representación:

- Se representan como una **línea continua**. Los mensajes "navegan" por las relaciones entre clases, es decir, los mensajes se envían entre objetos de clases relacionadas, normalmente en **ambas direcciones**.
- A veces la definición del problema hace necesario que se navegue en **una sola dirección**, entonces la línea finaliza en **punta de flecha**.
- Si no se indica navegabilidad, se tiende a pensar que es bidireccional.

Cardinalidad:

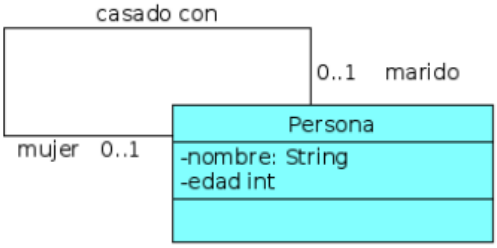
Las relaciones se caracterizan por su **cardinalidad**, que representa cuantos objetos de una clase se pueden involucrar en la relación.

Ejemplo: crea una clase nueva llamada Alumno y establece una relación de asociación con el nombre "matrícula" entre ésta y la clase Módulo.



Relaciones reflexivas:

Es posible establecer relaciones de una clase consigo misma.



Otros tipos de relaciones que se verán más adelante son:

- De generalización.
- De composición.
- De agregación.

6. Cardinalidad de una relación

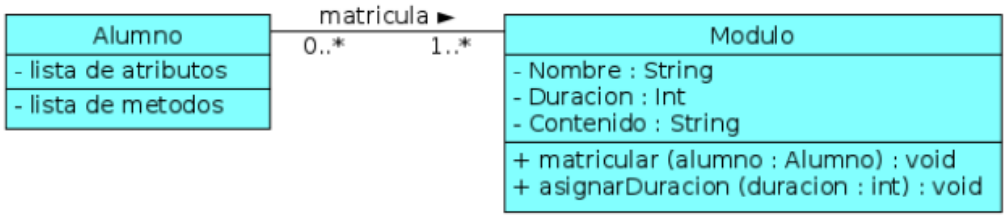
La **cardinalidad** de una relación, representa cuantos objetos de una clase se van a relacionar con objetos de otra clase. En una relación hay **dos cardinalidades**, una para cada extremo de la relación y pueden tener los siguientes valores:

Significado de las cardinalidades.

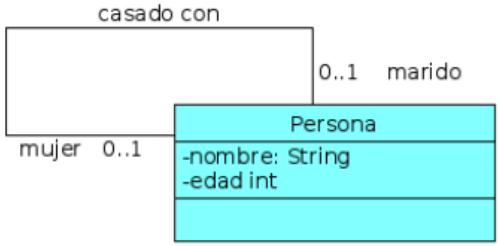
Cardinalidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

En las relaciones anteriores tendríamos:

- Un alumno tendrá que estar matriculado en al menos un módulo, pudiendo estar en varios y un módulo podrá estar sin alumnos o tener varios.



- Para la relación casado con, un marido podrá estar soltero o tener una mujer. La relación en la otra dirección tiene la misma cardinalidad.



7. Generalización (herencia)

La **generalización** es una propiedad que permite a los objetos ser construidos a partir de otros objetos, es decir, la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados. También recibe el nombre de **herencia**.

El objetivo principal de la generalización es la *reutilización*, poder utilizar código desarrollado con anterioridad. La herencia supone una clase base y una jerarquía de clases que contiene las clases derivadas. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes.

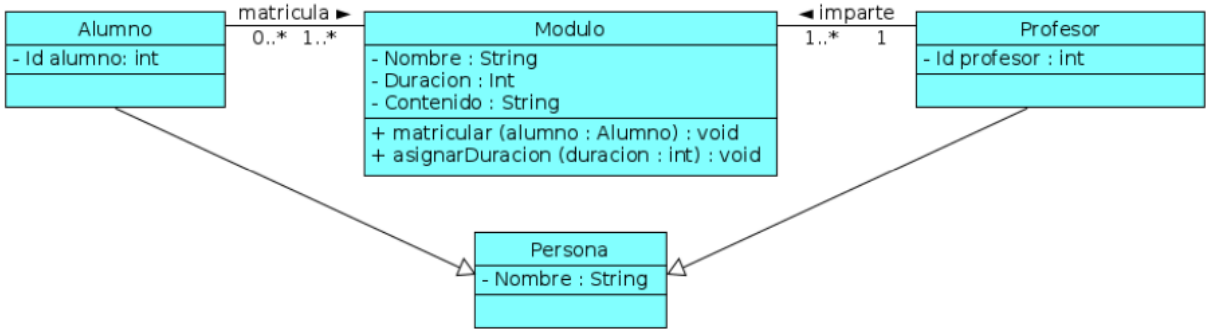
Tipos:

- **Herencia simple**: una clase puede tener sólo un ascendente inmediato. Es decir, una subclase puede heredar datos y métodos de una única clase base.
- **Herencia múltiple**: una clase puede tener más de un ascendente inmediato, adquirir datos y métodos de más de una clase.

Nota: algunos lenguajes como Java no soportan herencia múltiple.

Representación:

En el diagrama de clases se representa como una asociación en la que el extremo de la clase base tiene un triángulo.

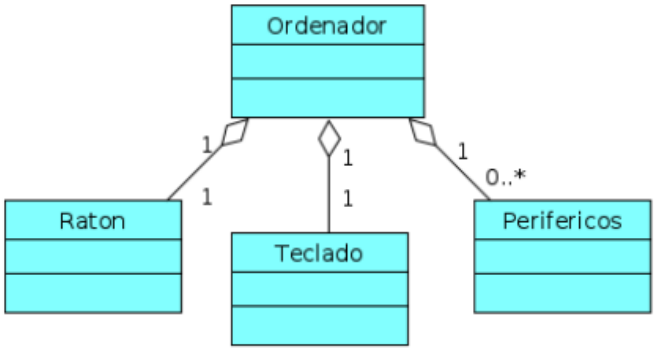


8. Agregación y composición

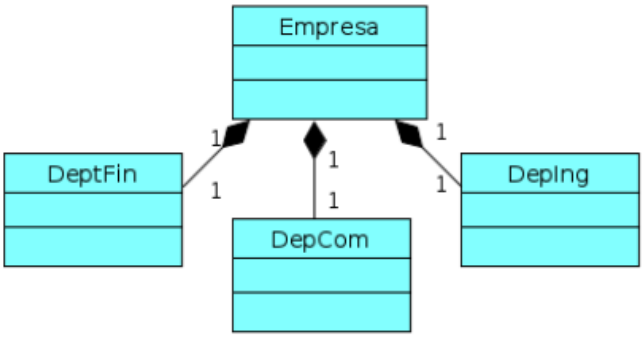
Muchas veces una determinada entidad existe como un conjunto de otras entidades. En este tipo de relaciones un objeto componente se integra en un objeto compuesto. La orientación a objetos recoge este tipo de relaciones como dos conceptos: **la agregación y la composición**.

- **La agregación** es una asociación binaria que representa una relación **todo-parte** (pertenece a, tiene un, es parte de). Los elementos *parte* pueden existir sin el elemento *todo* y no son propiedad suya. Por ejemplo, un centro comercial tiene clientes o un equipo tiene unos miembros. El tiempo de vida de los objetos no tiene porqué coincidir.

  En el siguiente caso, tenemos un ordenador que se construye a partir de piezas sueltas que pueden ser sustituidas y que tienen entidad por sí mismas, por lo que se representa mediante relaciones de agregación. Utilizamos la agregación porque es posible que una caja, ratón o teclado o una memoria RAM existan con independencia de que pertenezcan a un ordenador o no.



- **La composición** es una agregación fuerte en la que una instancia 'parte' está relacionada, como máximo, con una instancia 'todo' en un momento dado, de forma que cuando un objeto 'todo' es eliminado, también son eliminados sus objetos 'parte'. Por ejemplo: un rectángulo tiene cuatro vértices, un centro comercial está organizado mediante un conjunto de secciones de venta, una empresa está formada por departamentos ....



Estas relaciones se representan con un rombo en el extremo de la entidad contenedora (todo):

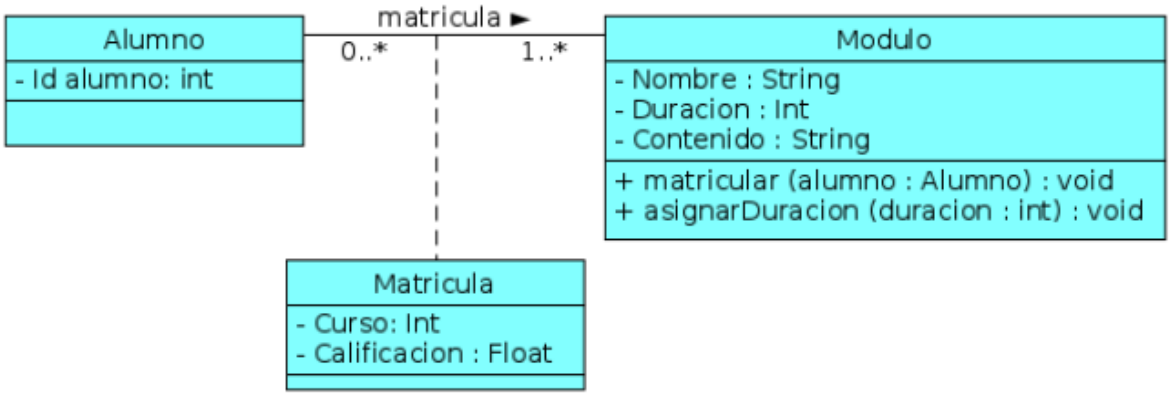
- En el caso de la agregación es de color blanco
- Para la composición **negro**.

Como en toda relación hay que indicar la cardinalidad.

9. Atributos de enlace

Es posible que tengamos alguna relación en la que sea necesario añadir algún tipo de información que la complete de alguna manera. Cuando esto ocurre podemos añadir **atributos a la relación**.

Ejemplo: cuando un alumno se matricula de un módulo es preciso especificar el curso al que pertenece la matrícula, las notas obtenidas en el examen y la tarea y la calificación final obtenida. Estas características no pertenecen totalmente al alumno ni al módulo sino a la relación específica que se crea entre ellos, que además será diferente si cambia el alumno o el módulo. Por tanto, la clase matrícula tiene una relación con el enlace entre Alumno y Módulo.



10. Restricciones

En ocasiones la relación entre dos clases está condicionada al cumplimiento de algún requisito, o un parámetro de una clase que tiene un valor constante ...

Cuando se precisa reflejar una condición que aparece en el enunciado y no disponemos de una notación particular para que quede reflejada en el diagrama de clases, es posible mostrarla mediante una restricción.

Las restricciones se incluyen mediante una descripción textual encerrada entre llaves.

- **Ejemplo-1**. Supongamos un ejercicio donde el socio de un club de fútbol desea acceder al palco del estadio. Si esta posibilidad sólo está disponible para antiguos jugadores del equipo, junto a la línea que relaciona las clases socio y palco, se puede incluir la restricción {sólo para exjugadores}.

- **Ejemplo-2**. Supongamos que la clase producto de un establecimiento tiene un IVA fijo del 21%, sea cual sea el tipo de producto que pone a la venta. Dado que este valor es fijo, podría considerarse una constante. En el diagrama de clases podría indicarse con una restricción del tipo {IVA contante 21%}.

11. Pautas para crear diagramas de clases

A la hora de crear diagramas de clases, la clave está en hacer una buena elección de las clases que sugiere el problema.

Para identificar las clases candidatas a formar parte del diagrama, es recomendable subrayar cada nombre o sintagma nominal que aparece en el enunciado.

Cuando tengamos la lista completa habrá que estudiar cada clase potencial para ver si, finalmente, es incluida en el diagrama. Para ayudarnos a decidir, podemos utilizar los siguientes criterios:

- La información de la clase es necesaria para que el sistema funcione.
- La clase posee un **conjunto de atributos** que podemos encontrar en cualquier ocurrencia de sus objetos. Si sólo aparece un atributo normalmente se rechazará y será añadido como atributo de otra clase.
- La clase tiene un **conjunto de operaciones** identificables que pueden cambiar el valor de sus atributos y son comunes en cualquiera de sus objetos.

- Es una entidad externa que consume o produce información esencial para la producción de cualquier solución en el sistema.

La clase se considera si cumple todos (o casi todos) los criterios.

Se debe tener en cuenta que **la lista obtenida no incluye todo**, habrá que añadir clases adicionales para completar el modelo y también, que diferentes descripciones del problema pueden provocar la toma de diferentes decisiones al seleccionar las clases y sus atributos.