

UML

Contenido

Introducción a UML

- ❖ ¿Qué es UML?
- ❖ Tipos de diagramas en UML.
- ❖ Tipos de relaciones entre clases.
- ❖ Herramientas de modelado.



¿Qué es UML?

UML (**U**nified **M**odeling **L**anguage) es un lenguaje de modelado visual de software, indispensable para la arquitectura y la ingeniería de software y sistemas.

Fue pensado y creado como una lengua universal para desarrolladores.

UML es el más conocido y utilizado en la actualidad y está respaldado por el OMG (**O**bject **M**anagement **G**roup).



Tipos de diagramas de UML

UML dispone de 13 tipos de diagramas dividido en 2 grandes categorías.

- ❖ **Diagramas de estructura.**
- ❖ **Diagramas de comportamiento.**

Diagramas de estructura

Este tipo de diagramas grafican la estructura estática de un sistema, sin información sobre el ciclo de vida de sus componentes.

Diagramas de estructura más comunes:

- ❖ **Diagrama de clases:** Muestra las clases intervinientes en un sistema y sus correspondientes relaciones.
- ❖ **Diagrama de paquetes:** Muestra los paquetes intervinientes en un sistema y sus correspondientes relaciones.
- ❖ **Diagrama de objetos:** Muestra el estado de los objetos intervinientes en un sistema en un determinado instante.

Diagramas de comportamiento

Este tipo de diagramas grafican la estructura dinámica de un sistema, la cual sufre cambios de estado o de componentes en el tiempo.

Diagramas de comportamiento más comunes:

- ❖ **Diagrama de casos de uso:** Muestra los diferentes actores intervinientes en un sistema junto a la funcionalidad requerida y sus interacciones.
- ❖ **Diagrama de secuencia:** Muestra la interacción en orden de los diferentes componentes de un sistema.

Diagrama de clases - Definición de clases

Una clase se dibuja como un rectángulo dividido en tres compartimientos horizontales:

- ❖ **Superior:** Lleva el nombre de la clase.
- ❖ **Central:** Lleva la lista de atributos.
- ❖ **Inferior:** Lleva la lista de constructores y métodos.

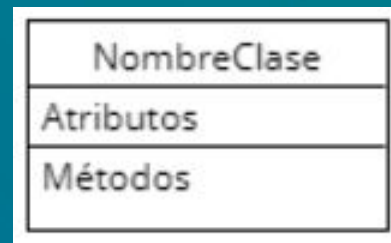


Diagrama de clases - Atributos

Se inserta en cada línea el nombre del atributo seguido de **:** y el tipo de dato.

Ejemplo:

En este caso, las personas tienen tres atributos: *nombre*, *apellido* y *DNI*.

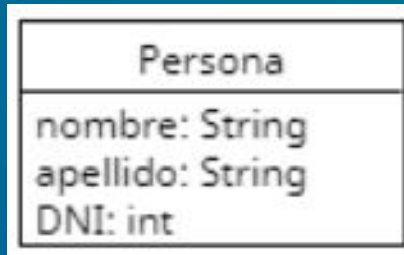


Diagrama de clases - Constructores y métodos

Los constructores se suelen poner al principio seguido de la lista de tipos de datos de sus parámetros entre paréntesis, separados por comas.

Si no hay parámetros, se dejan los paréntesis vacíos.

Para los métodos, se escribe el nombre del método, seguido de la lista de tipos de datos de sus parámetros entre paréntesis, separados por comas.

Si no hay parámetros, se dejan los paréntesis vacíos.

A continuación se coloca un `:` y el tipo de dato de retorno (si no hay retorno, se escribe `void`).

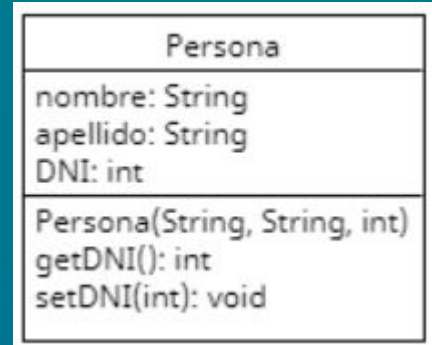




Diagrama de clases - Visibilidad

Cada uno de los miembros de una clase (atributos, constructores y métodos) puede tener diferente visibilidad, las cuales se modelan de la siguiente manera:

- ❖ **Público:** Se representa con un **+**. Indica que ese miembro es accesible desde cualquier lugar.
- ❖ **Privado:** Se representa con un **-**. Indica que ese miembro es accesible solo desde la propia clase.
- ❖ **Protegido:** Se representa con un **#**. Indica que ese miembro es accesible desde la propia clase, por otras clases dentro del mismo paquete y por sus subclases (sin importar el paquete donde se encuentren).
- ❖ **De paquete:** No se representa. Indica que ese miembro es accesible desde la propia clase y por otras clases dentro del mismo paquete.

Diagrama de clases - Visibilidad

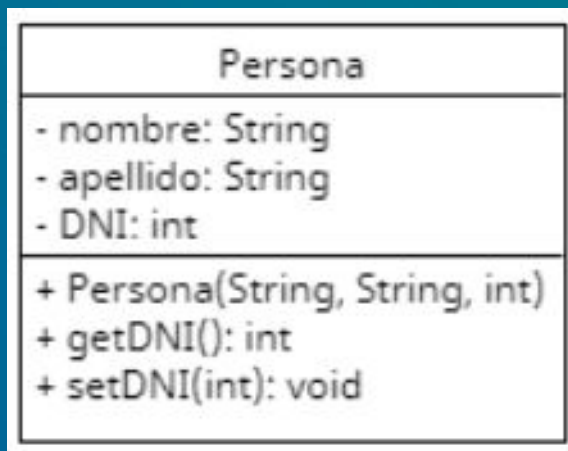


Diagrama de clases - Miembros estáticos

Los miembros **de clase** o **estáticos**, se los debe subrayar para diferenciarlos de los miembros de **instancia**, es decir, los **no** estáticos.

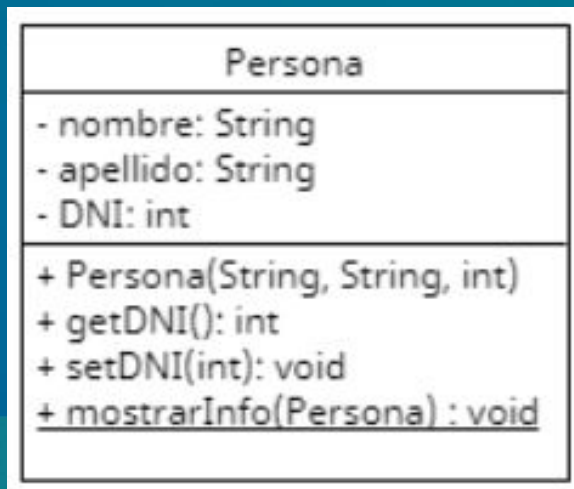


Diagrama de clases - Clases abstractas

Cuando un **método** o una **clase** sea **abstracto** se lo debe escribir en *itálica* para diferenciarlo con métodos o clases concretas.

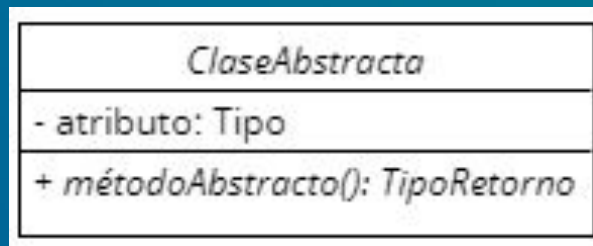


Diagrama de clases - Interfaces

Las interfaces son un tipo especial de clases abstractas que únicamente poseen *métodos abstractos* o *constantes de clase*.

Para diferenciarlas de las clases abstractas que podrían tener además atributos y métodos concretos, se utiliza el estereotipo **<<interface>>** justo arriba de su nombre.

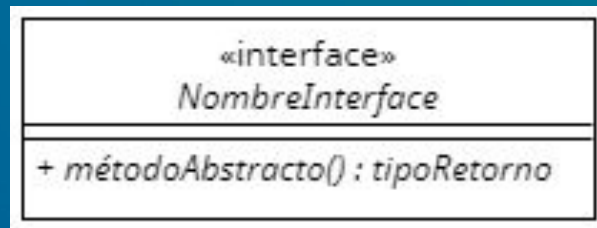
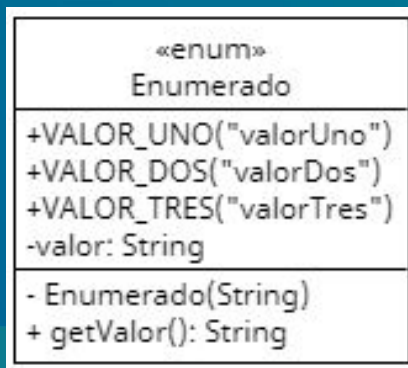


Diagrama de clases - Enumerados

Los enumerados son clases enumeradas, las cuales podrán tener constantes, atributos, métodos y constructor.

Para diferenciarlas de las clases, se utiliza el estereotipo **<<enum>>** justo arriba de su nombre.

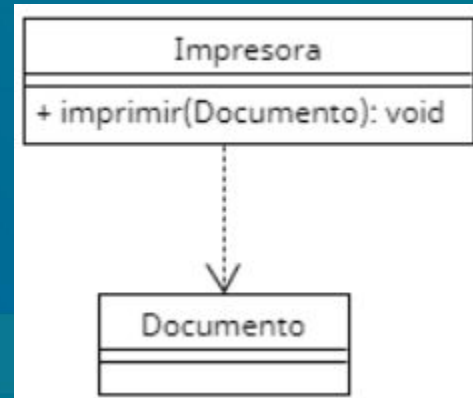
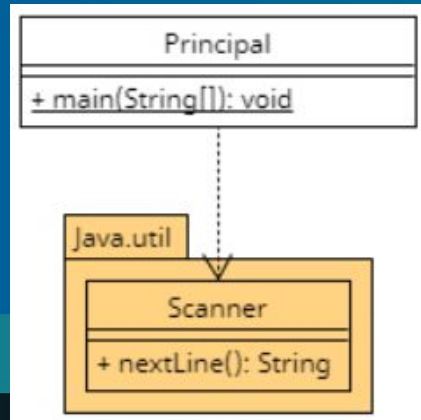


Tipos de relaciones entre clases

Dependencia

Cuando una clase utiliza a otra clase se habla de **uso** o **dependencia**. Es una relación débil, ya que la clase que utiliza **no** tiene un *atributo del tipo de la clase utilizada*, sino que simplemente instancia un objeto como variable local en un *método* o lo recibe por *parámetro*.

Una dependencia entre clases se representa con la siguiente flecha:

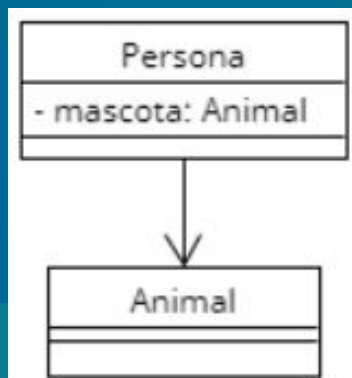


Asociación

Cuando una clase tiene un atributo cuyo *tipo* es de otra clase, hablamos de **asociación**.

Este tipo de relación puede tener cualquier significado y no especifica el ciclo de vida de los objetos intervinientes.

Una asociación entre clases se representa con la siguiente flecha:



Agregación

Un objeto puede estar compuesto de otro objeto. El ciclo de vida del objeto componente es **independiente** del objeto compuesto, esto significa que, si el objeto componente se destruye, el objeto compuesto de todas maneras puede existir (exista una referencia hacia él en otra clase).

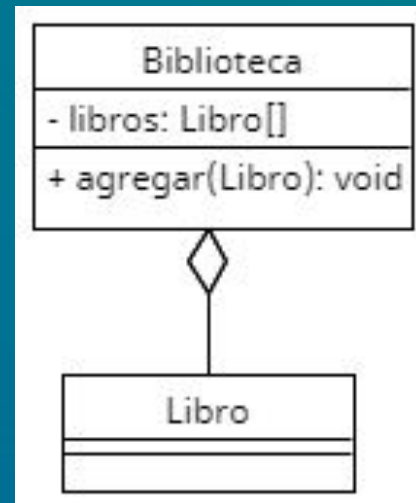
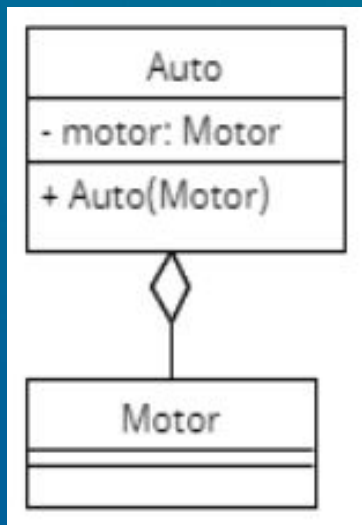
Este tipo de relación, se llama **agregación** y se cumple que un objeto **tiene un** (has-a) objeto.

Para que se cumpla esta relación, el objeto compuesto se recibe como parámetro (podría ser en un *constructor* o en un método *setter*). Esto asegura que tal objeto se creó en otro lugar, asegurando que exista una referencia a él si el objeto compuesto se destruye.

Una agregación entre clases se representa con la siguiente flecha:



Agregación



Composición

Un objeto puede estar compuesto de otro objeto. El ciclo de vida del objeto componente es totalmente **dependiente** del objeto compuesto, esto significa que, si el objeto componente se destruye, el objeto compuesto se destruye también (hay una única referencia hacia él en el objeto componente).

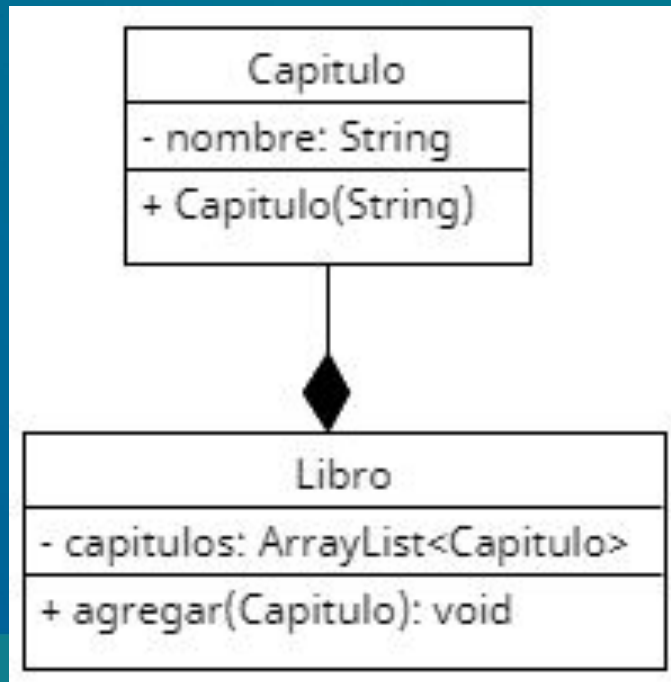
Este tipo de relación, se llama **composición** y se cumple que un objeto **tiene un** (has-a) objeto.

Para que se cumpla esta relación, el objeto compuesto se instancia dentro de la clase del objeto componente. Esto asegura que ambos tengan el mismo ciclo de vida.

Una composición entre clases se representa con la siguiente flecha:



Composición



Generalización o herencia

Cuando una subclase (también llamada *clase hija* o *clase derivada*) comparte estado y comportamiento con una *superclase* (también llamada *clase padre* o *clase base*), la relación es de **generalización**.

Las subclases heredan todos los atributos y métodos de la superclase que no sean privados.

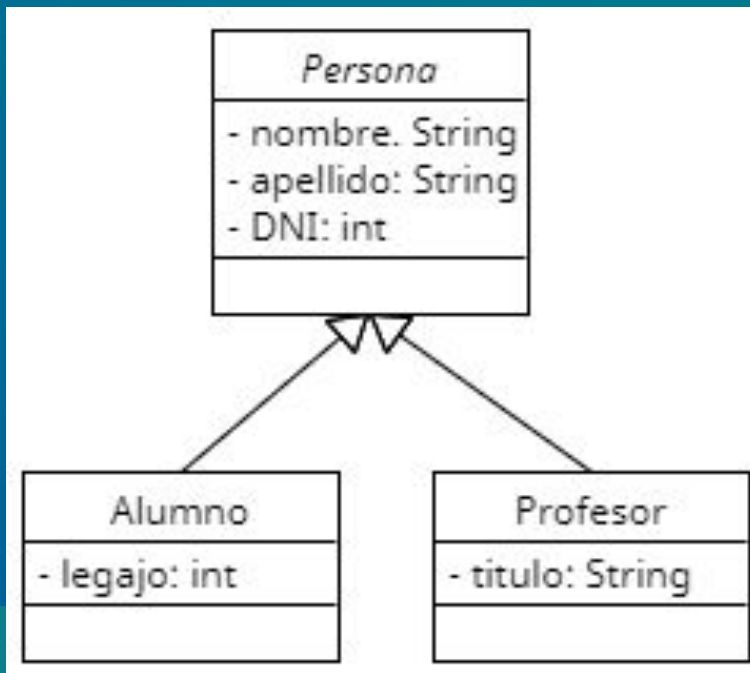
Los constructores **no** se heredan.

Se cumple que el objeto de la subclase **es un** (is-a) tipo especial de objeto de la superclase.

Una generalización entre clases se representa con la siguiente flecha:



Generalización o herencia



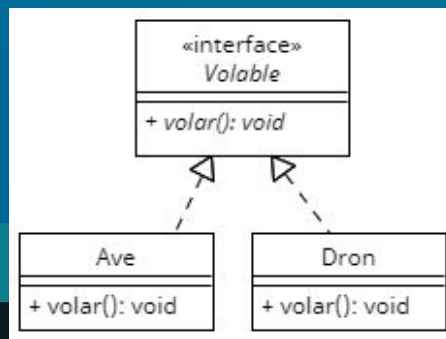
Realización

Cuando una clase *cumple* con un contrato especificado en una **interfaz**, la relación es de **realización**.




La clase debe sobrescribir todos los miembros de la interfaz, los cuales son públicos y abstractos.

En la realización se cumple también la relación **es un** (is-a).

Una realización entre clases e interfaces se representa con la siguiente flecha:



Resumen - Relaciones entre clases

Tipo de flecha	Nombre de la relación
	Dependencia
	Asociación
	Agregación
	Composición
	Generalización
	Realización

UMLetino

¿Qué es UMLetino?

UMLetino es una herramienta simple y gratuita que permite modelar rápidamente diagramas **UML** en la web.

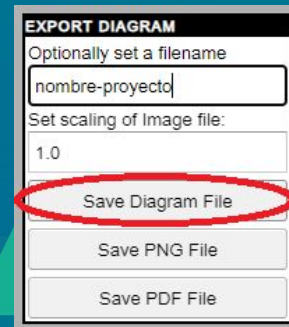
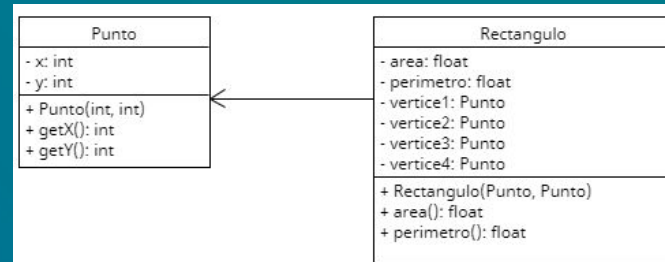
Se ejecuta enteramente en el navegador y no necesita ningún tipo de instalación.




¿Cómo integrarlo a la IDE?

Una vez finalizado el diagrama de clases en *UMLetino*, se deberán seguir los siguientes pasos:

- ❖ Pulsar *File Export*.
- ❖ Indicar nombre-del-proyecto.
- ❖ Pulsar el botón *Save Diagram File*.
- ❖ Indicar dónde se guardará el archivo **.uxf**



¿Cómo integrarlo a la IDE?

Acceder a  . Esta herramienta nos permitirá, a partir de un archivo **.uxf**, generar el código equivalente en el lenguaje de programación Java.

- ❖ Seleccionar el archivo **.uxf** previamente generado.
- ❖ Opcionalmente, se pueden configurar los tipos de parámetros no escalares.
- ❖ Se recomienda indicar el nombre del **package**.
- ❖ Seleccionar versión de JDK (no importa si es anterior a la que tenemos).
- ❖ Indicar para qué IDE se importará el proyecto.
- ❖ Por último, se tiene que pulsar el botón *Descargar código Java*. Se descargará un archivo **.zip**.



¿Cómo integrarlo a la IDE?

1) Cargá el diagrama de clases generado por el software [UMLet](#) o su versión online [UMLetino](#).

Seleccionar archivo

* Debe tener extensión ".uxf"

2) Los atributos para relaciones de cero a muchos serán de tipo...

☒ ArrayList ☐ LinkedList ☐ List ☐ Collection

3) Nombre del paquete que agrupará todas las clases...

Por ejemplo: `ar.charlycimino.proyectos`. Dejar vacío si no se desea un paquete (no recomendado).

4) Versión del JDK (Java Development Kit)...

▼

Las versiones de la lista son las recomendadas, por ser LTS (Long Time Support)

5) El proyecto generado será para importar en...

☒ NetBeans ☐ Eclipse

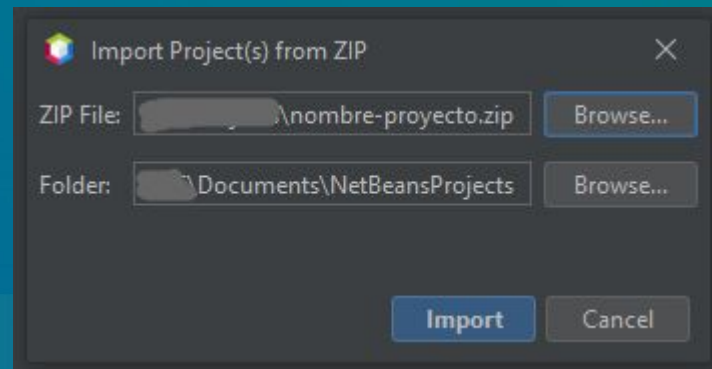
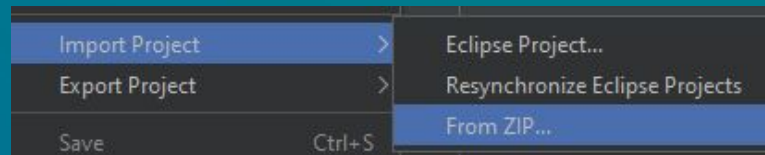
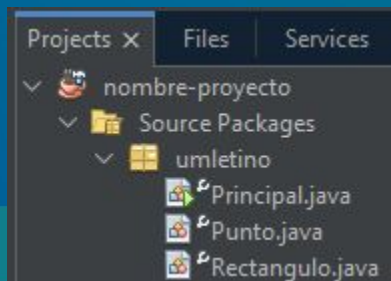
[Descargar código Java](#)



¿Cómo integrarlo a la IDE?

Una vez en NetBeans, se deberá importar el proyecto generado anteriormente, realizando los siguientes pasos:

- ❖ Acceder a la opción de importar proyectos desde un .ZIP.
- ❖ Seleccionar el archivo generado e importarlo.
- ❖ y... Listo!!!



Ejercitación