

Clase 12. PYTHON

Programación Orientada a Objetos I

Esta clase va a ser

- grabada

a

Objetivos de la clase



Conocer qué es la POO.



Diferenciar la POO de la programación tradicional.

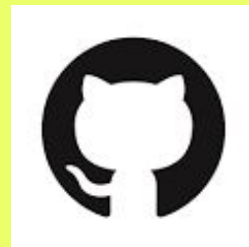


Aplicar el pensamiento computacional en un programa con objetos.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ [Repositorio Python](#)



Temario

11

Excepciones

- ✓ Errores
- ✓ Excepciones

12

Programación Orientada a Objetos I

- ✓ [¿Qué es la POO?](#)
- ✓ [Relaciones entre clases](#)

13

Programación orientada a objetos II

- ✓ Clases
- ✓ Atributos
- ✓ Métodos
- ✓ Clases anidadas
- ✓ Encapsulamiento

¿Qué es la POO?

¿Qué es la POO?

Es un modo o paradigma de programación, que nos permite organizar el código pensando el problema como una relación entre "cosas", denominadas objetos. Los objetos se trabajan utilizando las "clases". Estas nos permiten agrupar un conjunto de variables y funciones que veremos a continuación.

¿Que es la POO?

La Programación Orientada a Objetos (POO) es un paradigma de programación que utiliza la idea de "objetos" para organizar y estructurar el código.

¿Qué son los objetos?

- Los objetos son instancias de clases, que son prototipos para crear esos objetos.
- Cada objeto puede contener datos (también conocidos como atributos o propiedades) y funciones (métodos) que operan con esos datos.

¿Que es la POO?

Objeto: es una entidad que tiene dos características: estado y comportamiento.

Estado: está representado por los **atributos**, es decir las propiedades relevantes de un objeto. Los atributos de un objeto se almacenan en **variables de instancia**.

Comportamiento: está representado por una serie de **funciones o métodos** que modifican o no el estado del objeto.

Ejemplos:

Una **perro** tiene
estado: nombre, raza, color



comportamiento: ladrar, jugar, comer, etc.

Una **bicicleta** tiene
estado: cadena, nº de cambios, cambio actual
comportamiento: acelerar, frenar, etc.



Motivación

Motivación

Los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser reutilizados se creó la posibilidad de utilizar módulos. El primer módulo existente fue la función.

Pero la función se centra más en aportar una funcionalidad dada, pero no tiene tanto interés con los datos. 📌

Motivación

Con la POO se busca resolver aplicaciones cada vez más complejas, sin que el código se vuelva un caos. Además, se pretende dar pautas para realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que **el código se pueda reutilizar**.



En resumen...

Lo importante de la POO es poder separar los problemas generales en **suma de pequeños problemas aislados**, para poder modelar la solución y en un futuro que cualquiera pueda utilizar varios de estos módulos creados por este paradigma.

Diferencias con la programación tradicional

¿Qué es un paradigma de programación?

Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un **modelo para resolver problemas computacionales**.

Los lenguajes de programación se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.

Paradigma de programación

- ✓ **Imperativo:** Los programas se componen de un conjunto de sentencias que cambian su estado. Son secuencias de comandos que ordenan acciones a la computadora.

Por ejemplo describir los pasos para sumar los elementos de una lista

```
suma = 0
```

```
lista = [0,1,2,3]
```

```
for valor in lista:
```

```
    suma += valor
```


Paradigma de programación

- ✓ **Declarativo:** Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.

Un ejemplo de programación declarativa puede ser un programa en un lenguaje funcional que describe qué debe calcular en lugar de cómo hacerlo. (Haskell)

Paradigma de programación

-- Definir una función para calcular la suma de una lista de números

suma :: [Int] -> Int

suma lista = foldr (+) 0 lista

-- Lista de números

numeros :: [Int]

numeros = [1, 2, 3, 4, 5]

-- Imprimir el resultado

main :: IO ()

main = putStrLn \$ "La suma es: " ++ show (suma numeros)

Paradigma de programación

✓ **Lógico:** El problema se modela con enunciados de lógica del primer orden.

La programación lógica se basa en la lógica formal y la inferencia. Un ejemplo clásico de programación lógica es el lenguaje Prolog.

La programación lógica se centra en declarar hechos y reglas, y luego realizar consultas sobre esos hechos y reglas para obtener respuestas lógicas. La inferencia lógica se encarga de encontrar soluciones que cumplan con las reglas establecidas.

es_hijo(Juanito, Juan)
suma(10,5,15)

-----> *Juanito es hijo de Juan*

?es_hijo(x, Juan)

Paradigma de programación

- ✓ **Funcional:** Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.
- ✓ **Orientado a Objetos:** El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema a resolver y tienen atributos y comportamiento.

Paradigma de programación Python

- ✓ **Imperativo:** Python permite la programación imperativa, donde se describen los pasos específicos que la computadora debe seguir para realizar una tarea. Esto incluye la asignación de variables, bucles, y condicionales.
- ✓ **Orientado a Objetos:** Python es un lenguaje orientado a objetos, lo que significa que permite la definición y el uso de clases y objetos. Las clases pueden tener atributos y métodos, y se pueden heredar y extender.
- ✓ **Procedural:** Además de ser imperativo, Python también admite la programación procedural. Puedes organizar tu código en procedimientos y funciones que se ejecutan secuencialmente.
- ✓ **Dinámico y Tipado Dinámico:** Python es un lenguaje de tipado dinámico, lo que significa que las variables no están asociadas a un tipo de datos específico durante la compilación. Esto brinda flexibilidad pero también puede llevar a errores en tiempo de ejecución si no se maneja adecuadamente.

Diferencias con la programación tradicional

Programación estructurada

- ✓ Énfasis en la transformación de datos.
- ✓ Las funciones y los datos son manejados como entidades separadas.
- ✓ Difícil de entender y modificar.

Programación orientada a objetos

- ✓ Énfasis en la abstracción de datos.
- ✓ Las funciones y los datos son encapsulados en una entidad.
- ✓ Facilita su mantenimiento y su comprensión es orientada al mundo real.

Ventajas de la programación orientada a objetos

Ventajas



No se encapsulan los atributos, ni el acceso a los atributos desde las funciones, que también están encapsuladas dentro de la clase.



Ofrece la posibilidad de heredar de unas clases a otras para que puedan acceder a los miembros de la clase padre, con lo cual podemos solucionar de una manera más eficiente, el ampliar el comportamiento de nuestros objetos o clases.

Ventajas



Podemos hacernos visualmente una idea más clara de cómo se puede comportar la clase, además de tener en el mismo sitio tanto las funciones que hacen que podamos manejar la clase como los datos que vamos a manejar, etc.



Break

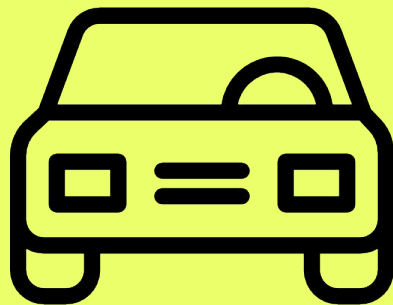
¡10 minutos y volvemos!

**¿Cómo se piensa
con POO?**

¿Cómo se piensa con POO?

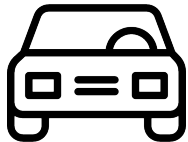
Es muy parecido a cómo lo haríamos en la vida real.
Por ejemplo: vamos a pensar en un coche para tratar de modelizar en un esquema de POO.

- ✓ **Elemento principal:** coche.
- ✓ **Características:** marca, modelo, color, tipo_motor.
- ✓ **Funcionalidades:** reversa, aparcamiento, consumo, aceleracion_0_100



¿Cómo se piensa con POO?

Ahora si pensamos el ejemplo anterior desde el esquema de Programación Orientada a Objetos, sería de la siguiente manera:



Elemento principal	_____•	Clase
Características	_____•	Atributos
Funcionalidades	_____•	Métodos

¿Cómo se piensa con POO?

Para poder comentar el anterior esquema de POO fácilmente aparecen los denominados **diagrama de clases**, que ilustran todo según estándares de la UML (Lenguaje Unificado de modelado).

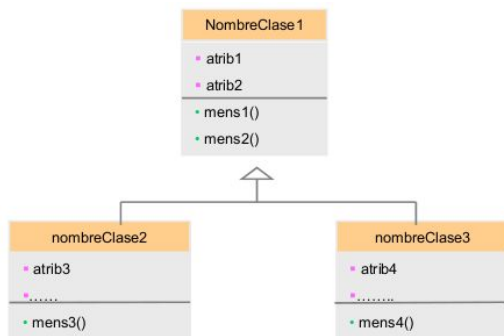


Diagrama de clases

Un diagrama de clase, es un diagrama de estructura estática que describe la estructura de un sistema.

Permite modelar sus clases, atributos, operaciones y relaciones entre objetos.

Un diagrama de clases describe:

- Los tipos de objetos en el sistema.
- Las relaciones estáticas que existen entre ellos.
- Los atributos y operaciones de las clases.
- Las restricciones a las clases y a sus asociaciones.

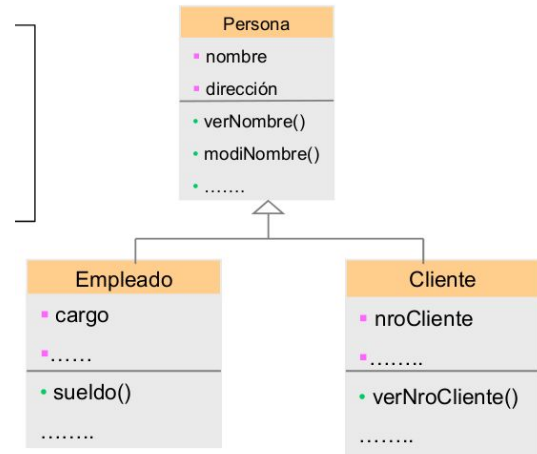
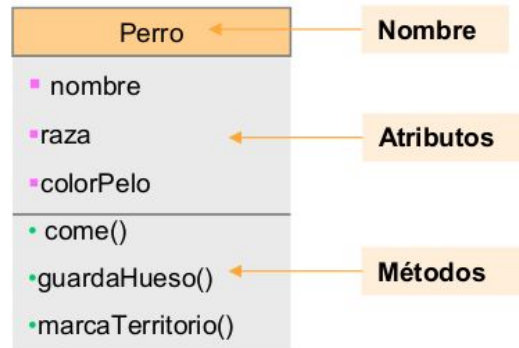


Diagrama de clases

Componentes básicos de un diagrama de clases

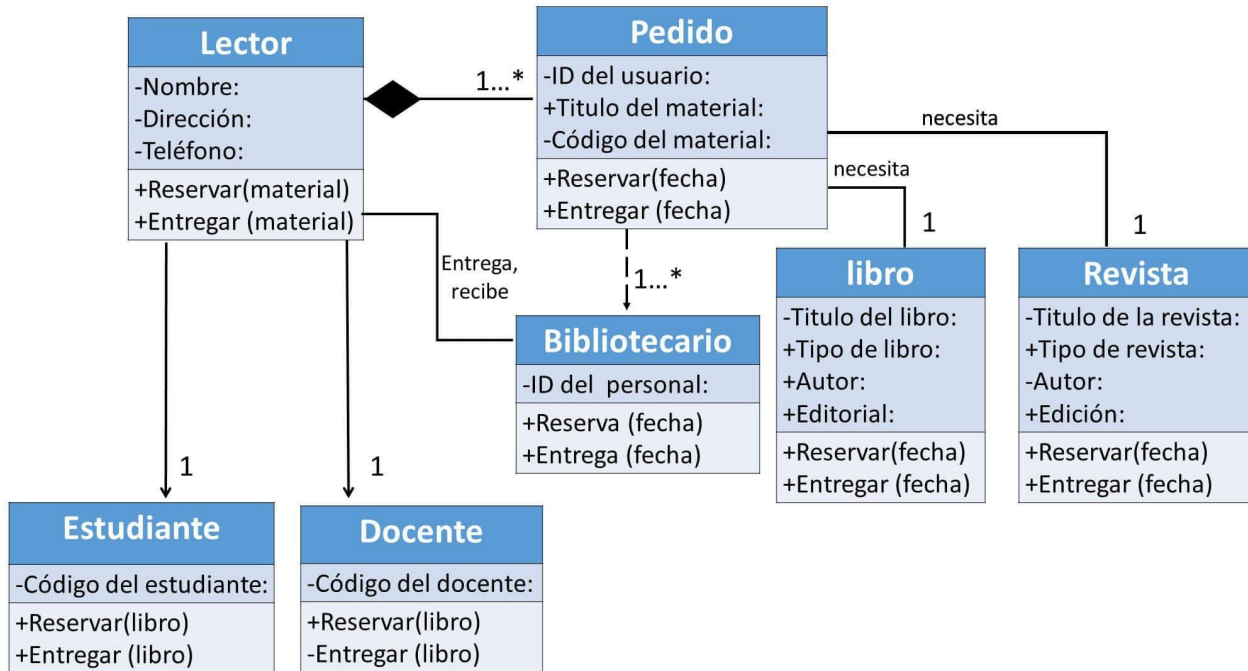
En UML las clases se representan mediante un rectángulo que puede estar dividido en tres partes.

- **Sección superior:** Contiene el nombre de la clase.
- **Sección central:** Contiene los atributos de la clase.
- **Sección inferior:** Incluye operaciones de clases (métodos), organizadas en un formato de lista. Cada operación requiere su propia línea.



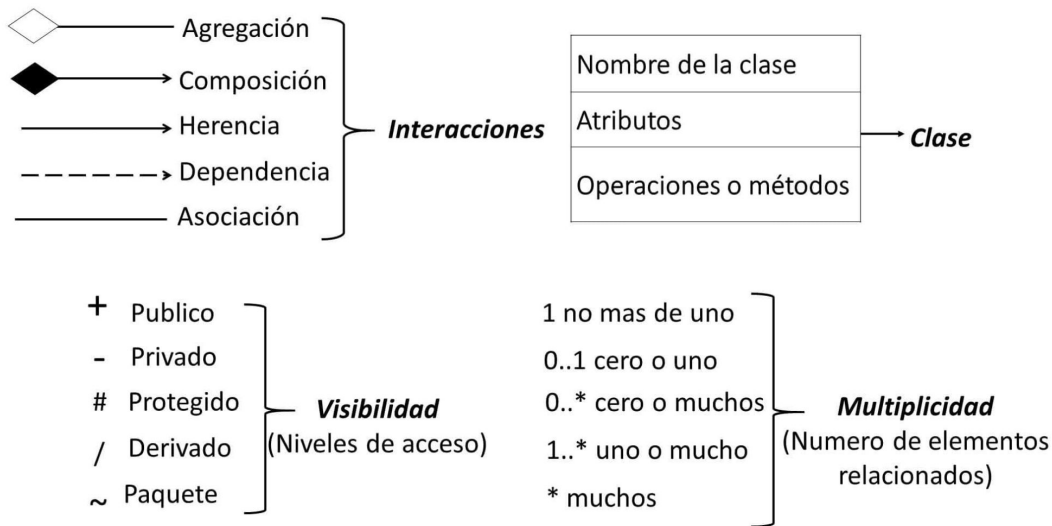
Ejemplo de un diagrama de clases

Diagrama de clases de un sistema de servicios bibliotecarios



Ejemplo de un diagrama de clases

Elementos y símbolos en los diagramas de clases UML



Con el ejemplo anterior notamos que los problemas complejos no son sólo una suma de Clases, son además **relaciones entre clases.**

Relaciones entre clases

Tipos de relaciones

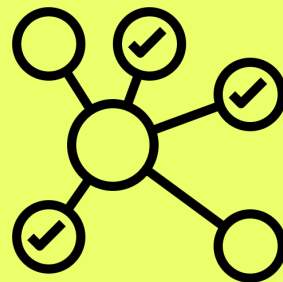
Tipos de relaciones

La Programación Orientada a Objetos (POO) intenta modelar aplicaciones del mundo real tan fielmente como sea posible y, por lo tanto, debe reflejar estas relaciones entre clases y objetos.

Las relaciones existente entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí.

Existen tres tipos de relaciones:

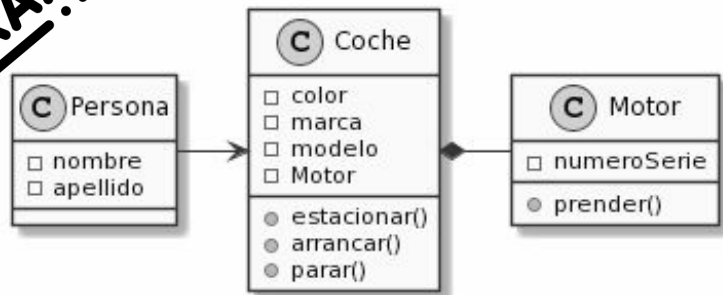
- ✓ Asociación
- ✓ Agregación / Composición
- ✓ Generalización / Especialización. Herencia simple y herencia múltiple.



1. Asociación

La asociación es una relación entre dos o más clases que permite que un objeto de una clase esté relacionado con un objeto de otra clase.

EXAMPLE



Especifica una relación semántica entre objetos no relacionados.

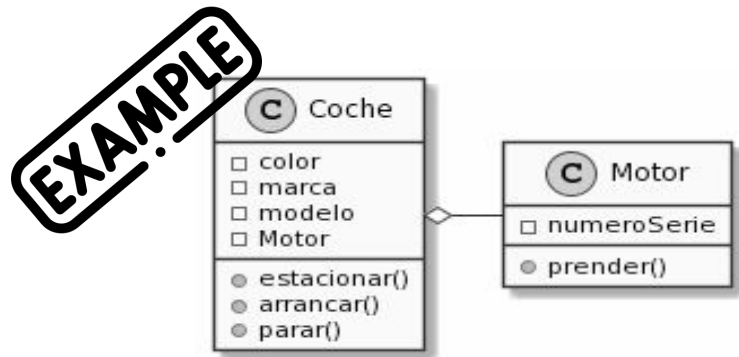
Este tipo de relaciones permiten crear asociaciones que capturan los participantes en una relación semántica.

Son relaciones del tipo **"pertenece_a"** o **"está_asociado_con"**.

Se da cuando una clase usa a otra clase para realizar algo.

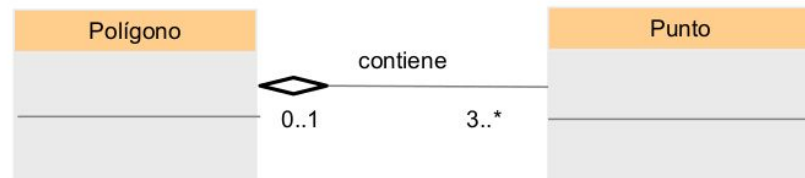


2. Agregación



Esta relación se presenta entre una clase **TODO** y una clase **PARTE** que es **componente de TODO**.

La implementación de este tipo de relación se consigue definiendo como atributo un objeto de la otra clase que es parte-de. Los objetos de la clase **TODO** son objetos contenedores. **Un objeto contenedor es aquel que contiene otros objetos.**

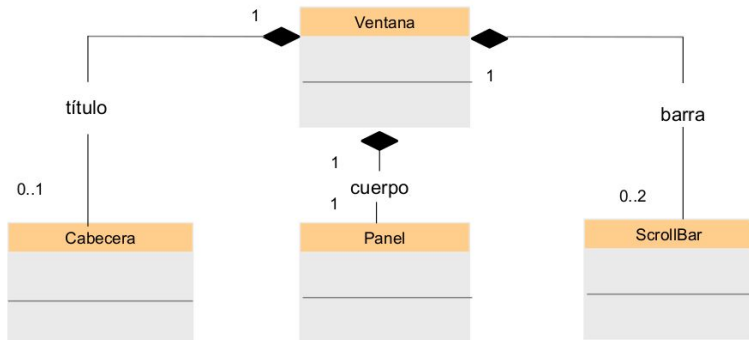
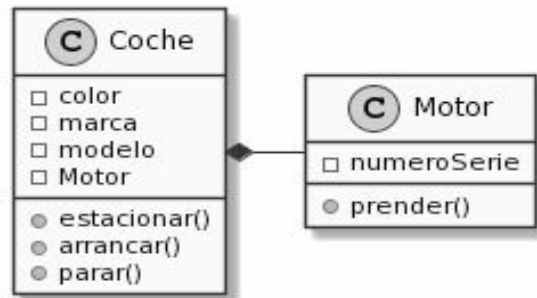


Las partes pueden formar parte de distintos agregados.

2. Composición

En una relación de composición, una clase se compone de otras clases, y las instancias de las clases componentes no pueden existir independientemente de la clase contenedora. La relación de composición es más fuerte que la agregación y generalmente se representa visualmente con un diamante sólido en los diagramas de clases.

EXAMPLE



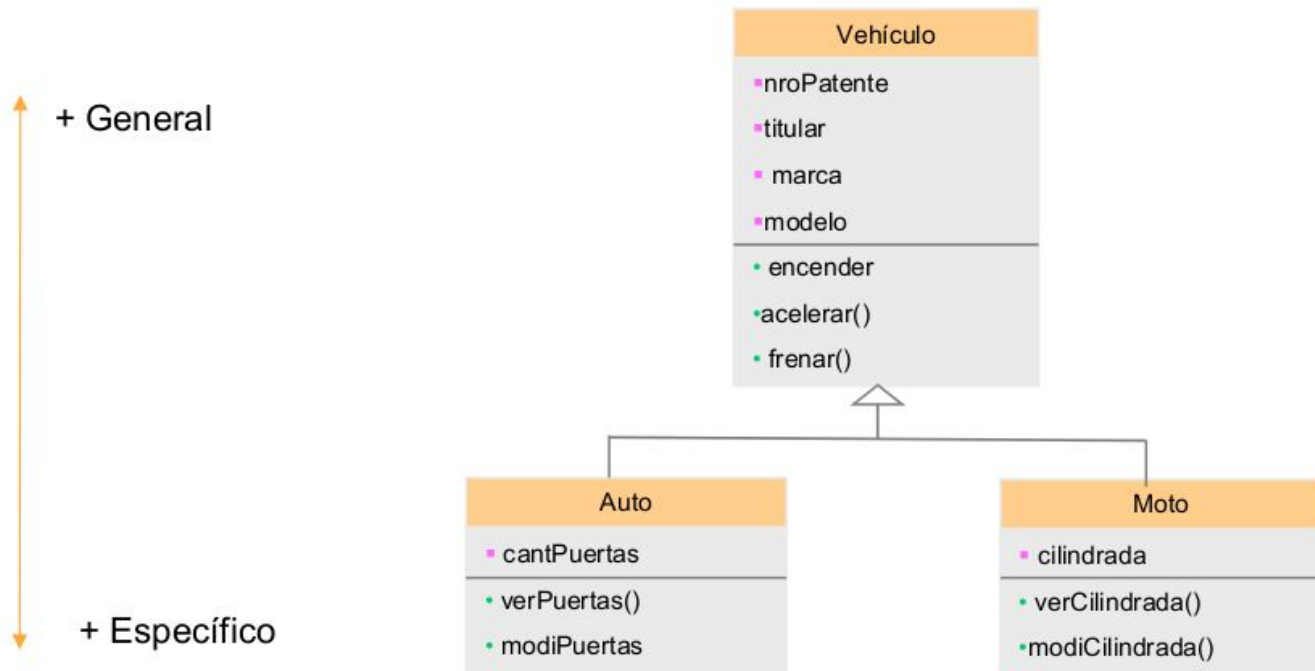
3. Generalización

De todas las relaciones posibles entre las distintas clases y objetos, hay que destacar por su importancia en la POO el concepto de herencia. La relación de herencia es una relación entre clases que comparten su estructura y el comportamiento.

Resulta importante destacar que esta temática será abordada en próximos encuentros, por el momento únicamente definiremos los siguientes conceptos:

- ✓ **Herencia simple:** Una clase comparte la estructura y comportamiento de una sola clase.
- ✓ **Herencia múltiple:** Una clase comparte la estructura y comportamiento de varias clases.

3. Generalización

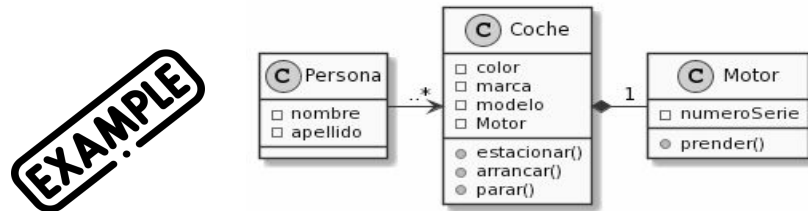


Cardinalidad de las relaciones

Cardinalidad en las relaciones

Sabemos que un coche tiene un motor, y que la persona está asociada a un vehículo. Ahí nace el **concepto de cardinalidad**, es decir indicar el número de Objetos que están en la relación. Por ejemplo, una persona puede tener muchos coches, y que los coches solo tienen un motor.

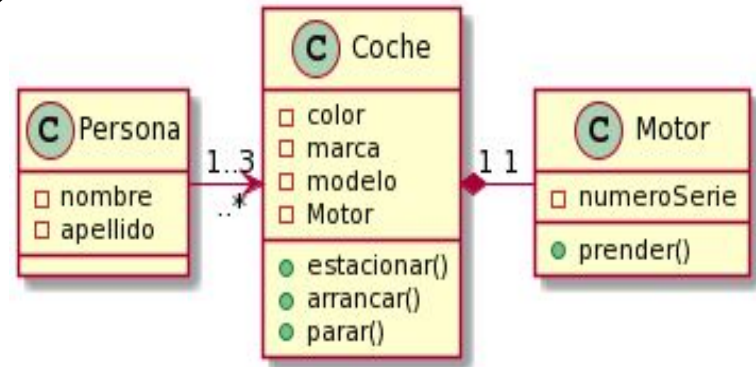
Además, sabemos que cada motor pertenece a un solo coche, y los coches a veces tienen más de un titular que lo usan, es decir tienen a varias personas para usarlo.



Cardinalidad en las relaciones

Solo por completitud del tema, y un poco alejado de la realidad, supongamos que un coche solo puede tener entre 1 y 3 titulares.

EXAMPLE



En la cardinalidad, lo importante es marcar si la relación es a muchos objetos ("**..***") o si solo es a uno (**1**), incluso a veces se puede decir a uno o ninguno (**0,1**).



Para pensar

Con lo aprendido hasta el momento en la clase están listos para responder ¿qué son los diagramas de clases?

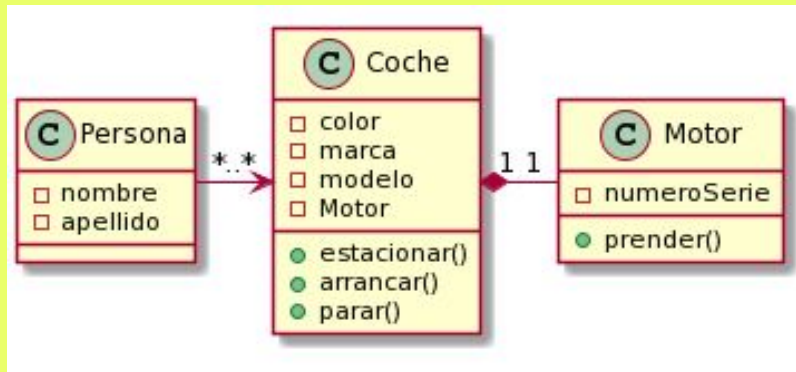
Contesta mediante el chat de Zoom

Diagrama de clases

¿Qué son?

Podemos definir a los diagramas de clases como las clases que usaremos en el programa con las relaciones entre sí. La siguiente imagen representa un diagrama de clases:

EXAMPLE



Graficar diagrama de clases

Herramientas para graficar diagrama de clases

Hay múltiples programas que nos permiten realizar los diagramas, los más populares son [Día](#), [Visio](#) y algunos otros online, como [Moqups](#) u [Online Visual Paradigm](#).

Nosotros usaremos [Drawing](#), una herramienta sumamente potente para la creación de diferentes esquemas y diagramas de sistemas.



Ejemplo en vivo







Vamos a ingresar a [DrawIO](#) y aprenderemos cómo usarlo.



Empecemos

Seleccionamos la opción de **Dispositivo**

Guardar diagramas en:

 Google Drive	 OneDrive	 Dispositivo
 Dropbox	 GitHub	 GitLab

Decidir más tarde



Empecemos

Creamos un
Nuevo Diagrama



Dispositivo



Crear nuevo diagrama

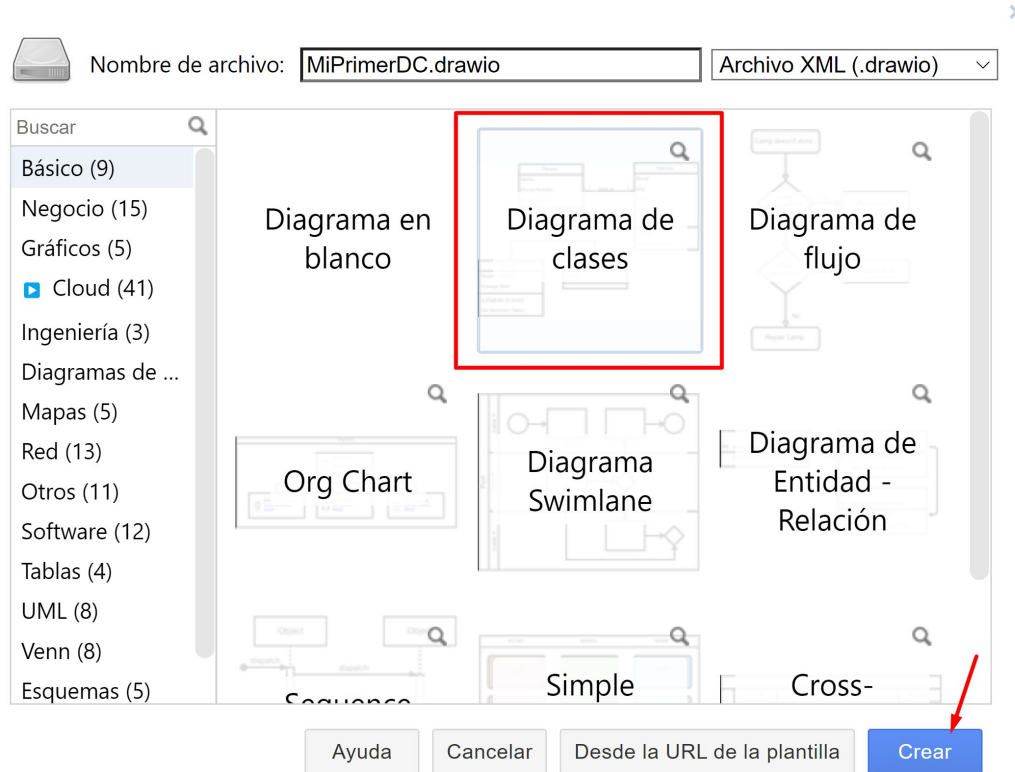
Abrir diagrama existente

Cambiar almacenamiento



Empecemos

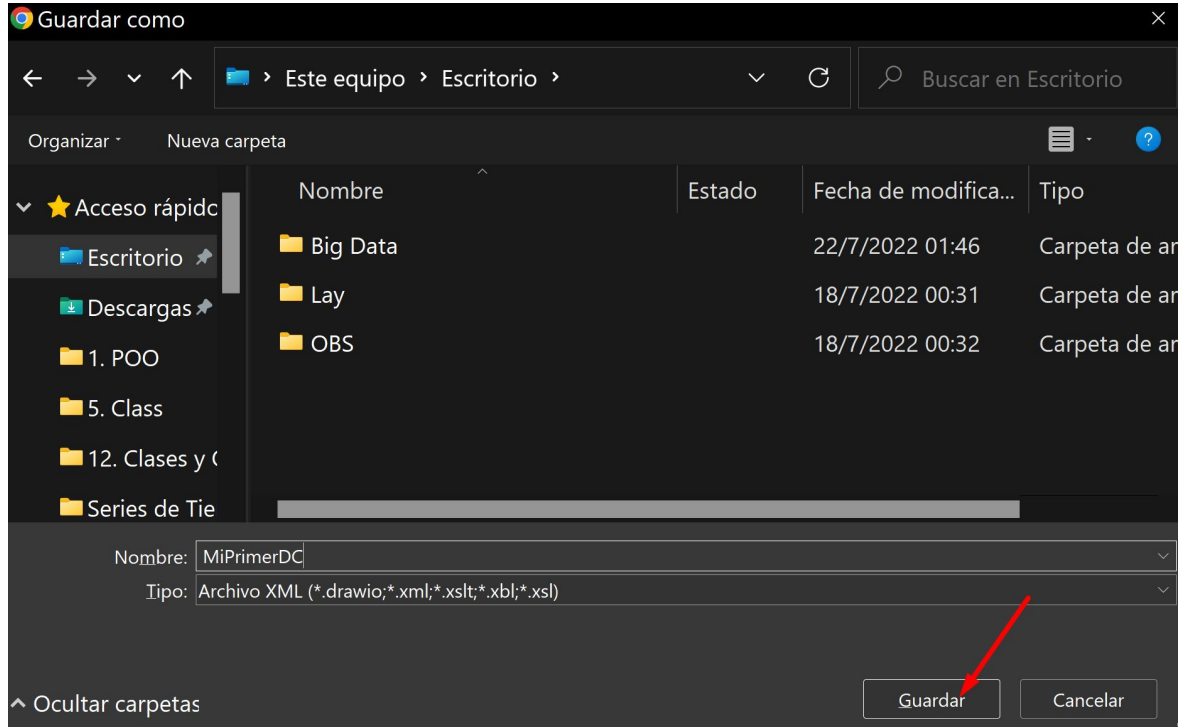
Seleccionamos
la opción de
**Diagrama
de Clases**





Empecemos

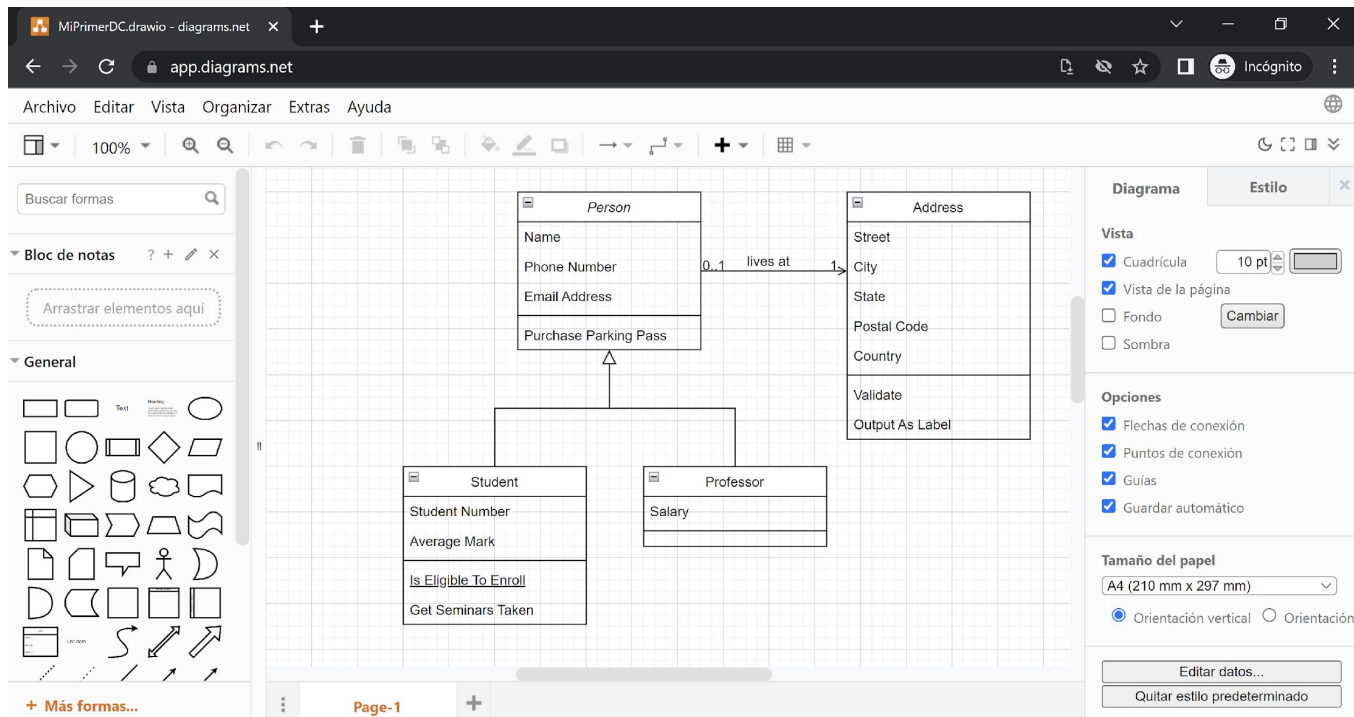
Elegimos una ruta en nuestro ordenador, por ejemplo: el **Escritorio**





Empecemos

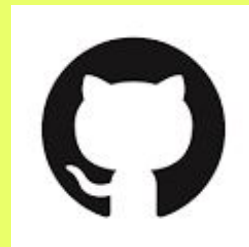
¡Listo!
ahora podemos
borrar todos
aquellos
componentes
que no
necesitemos



Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ [Repositorio Python](#)





Nuestro primer DC

Crear un diagrama de clases utilizando DrawIO.

Duración: **30 minutos**



ACTIVIDAD EN CLASE

Nuestro primer DC

Representa mediante un diagrama de clases el siguiente escenario:

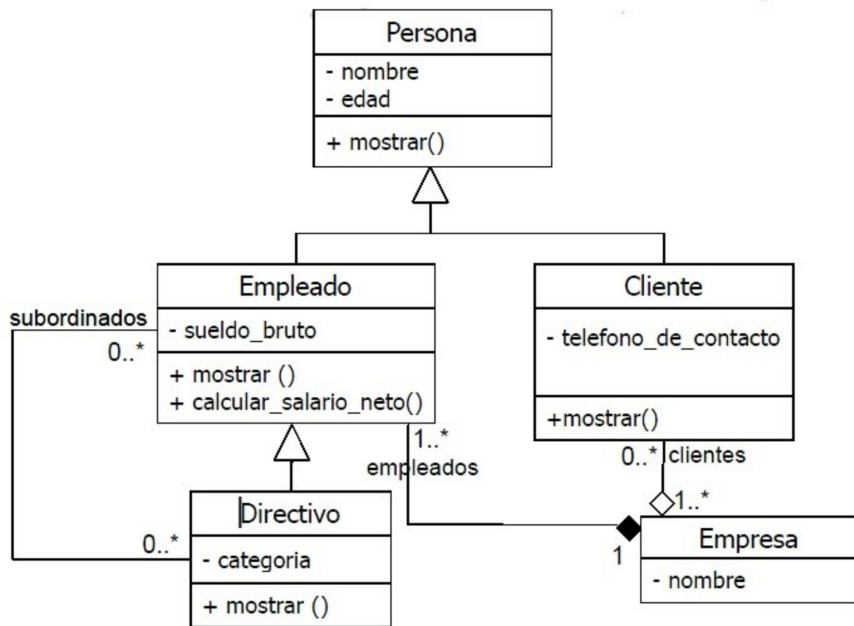
Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes. Ambos se caracterizan por su nombre y edad y derivan de una clase llamada Persona.

- ✓ Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.
- ✓ De los clientes además se necesita conocer su teléfono de contacto.
- ✓ La aplicación necesita mostrar los datos de empleados y clientes.



ACTIVIDAD EN CLASE

Nuestro primer DC





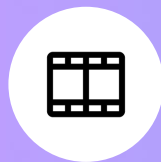
Ejemplo en vivo

¡Por último! Veamos un pequeño ejemplo de POO en Python.

Importante: El ejemplo que mostraremos a continuación es únicamente introductorio, en próximas sesiones se profundizará en la temática expuesta.

Notebook: *Clases_en_Python.ipynb*

¿Preguntas?



¿Quieres saber más?
**Te dejamos material
ampliado de la clase**



MATERIAL AMPLIADO

Recursos multimedia



[Ejemplo Clase](#)

Resumen de la clase hoy

- ✓ Programación Orientada a Objetos.
- ✓ Relaciones entre clases.
- ✓ Diagrama de clases.
- ✓ Diferencia entre POO y Tradicional.
- ✓ Ventajas de la POO.

Opina y valora
esta clase

Muchas gracias.

#DemocratizandoLaEducación