Programación Aplicada

FACULTAD DE INGENIERÍA Y NEGOCIOS

(*ACI222*)

Melissa Alegría Arcos, PhD.
malegriaa@udla.cl

2 de Abril del 2022

Contenido

5.4 Contenido: Laboratorio	
N° Unidad	**Tema**
1 Diseño de programa.	**La tarea de programar:** • El diseño antes que la programación. • El desafío de programar en equipo. **Cómo garantizar que el programa funcione: diseño de pruebas** • Elementos del paradigma de orientación a objetos que facilitan la modelación y programación: Herencia, Encapsulamiento, Polimorfismo.
2 Lenguajes de programación orientada a objetos	**Lenguajes de programación orientada a objetos:** • Java, C#, C , otros. **Python** • Contextos de aplicación. • Evaluación y selección de herramientas.
3 Lenguajes de programación multiparadigma	**Lenguajes de programación multiparadigma:** • ¿Para qué usar programación multiparadigma?. • Javascript, Python, PhP, otros. • Contextos de aplicación. • Evaluación y selección de herramientas. • Ejemplos en contextos reales.

Programación Orientada a Objetos POO

Según la revista informática (2016), Los lenguajes de programación orientados a objetos son todos aquellos lenguajes cuya sintaxis y gramática es soportada por este paradigma. Entre otros, tenemos:

Los principales lenguajes de programación orientados a objetos son:

Ada, C++, C#, VB.NET, Clarion, Delphi, Eiffel, Java, Lexico (en castellano), Objective-C, Ocaml, Oz, PHP, PowerBuilder, Python, Ruby y Smalltalk.

Los lenguajes de programación orientados a objetos tratan a los programas como conjuntos de objetos que se ayudan entre ellos para realizar acciones. Entendiendo como objeto al entidades que contienen datos.

La programación orientada a objetos puede tanto estar caracterizada por la siguiente sentencia:

Características basadas en objetos + herencia + ligadura dinámica

Tienen su origen en un lenguaje que fue diseñado por los profesores Ole-Johan Dahl y Kristen Nygaard en Noruega. Este lenguaje de programación orientado a objetos fue el "Simula 67" que fue un lenguaje creado para hacer simulaciones de naves.

Un nuevo paso en los lenguajes de programación es la Programación orientada a aspectos (POA). Actualmente esta en fase de desarrollo, pero cada vez atrae a más investigadores y empresas de todo el mundo.

No todos estos lenguajes de programación orientados a objetos son específicamente orientados a objetos. Sino que algunos de ellos se le han añadido extensiones orientadas a objetos.

Paradigma Orientado a Objetos

Las propiedades más importantes de la POO son:

- Abstracción.
- Encapsulamiento
- Herencia.
- Polimorfismo.

PPO (programación orientada a objetos) es un paradigma de programación.
Nos debemos preguntar ¿Qué objetos del mundo real se puede modelar?

Este paradigma esta basada en la encapsulación del estado y el comportamiento
Estado → con atributos y comportamiento → con métodos

- Se divide el sistema en modelos de objetos físicos o modelados y cada uno de estos objetos tiene un estado.

POO

Todo lo que veremos a continuación es el mismo material que esta en estas dos fuentes:
https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html
https://diagramasuml.com/diagrama-de-clases/

Conceptos clave

- Abstracción
- Clase
- Atributo
- Método
- Objeto
- Instancia
- Diagramas de clases
- Mensaje
- Constructor
- Destructor
- Evento
- Sobreescritura
- Clase Abstracta
- Interfaz
- Métodos de acceso
- Modificadores de acceso
- Modularidad
- Encapsulamiento
- Herencia
- Polimorfismo
- Atributos estáticos
- Métodos estáticos
- Clases estáticas
- Hilos

Abstracción

La abstracción es un proceso de interpretación y diseño que implica reconocer y enfocarse en las características importantes de una situación u objeto, y filtrar o ignorar todas las particularidades no esenciales.

- Dejar a un lado los detalles de un objeto y definir las características específicas de éste, aquellas que lo distingan de los demás tipos de objetos.
- Hay que centrarse en lo que es y lo que hace un objeto, antes de decidir cómo debería ser implementado.
- Se hace énfasis en el qué hace más que en el cómo lo hace

Clase

Nombre	Ave
Atributos Nombre Pico Color plumaje	
Métodos Cantar Comer Volar	

Objetos

Ave1
Nombre: Cardenal Pico: Corto Color: rojiso Plumaje: Espeso
Cantar: Silvido Comer: Insectos Volar: bajo

Ave2
Nombre:Trochilidae Pico: Largo Color: verdoso Plumaje: Fino
Cantar: Silvido Comer: Flores Volar: alto

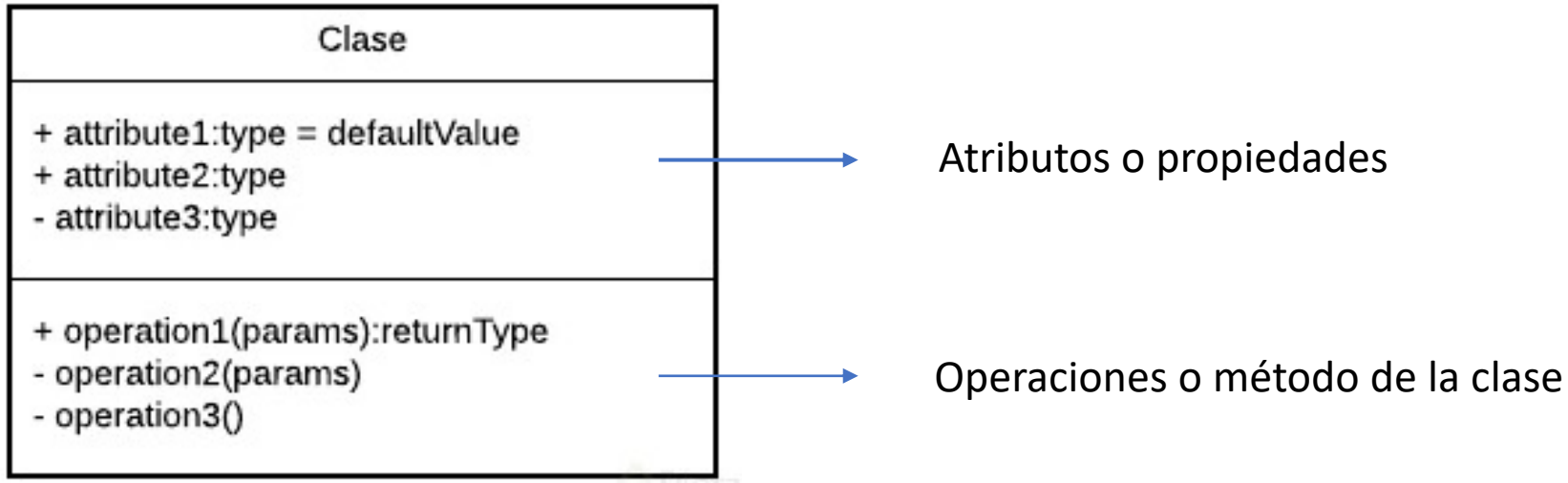
Clases, Propiedades, Métodos y Objetos

Una clase es una entidad que define una serie de elementos que determinan un estado (datos) y un comportamiento (operaciones sobre los datos que modifican su estado).

Un objeto es una instancia de una clase.

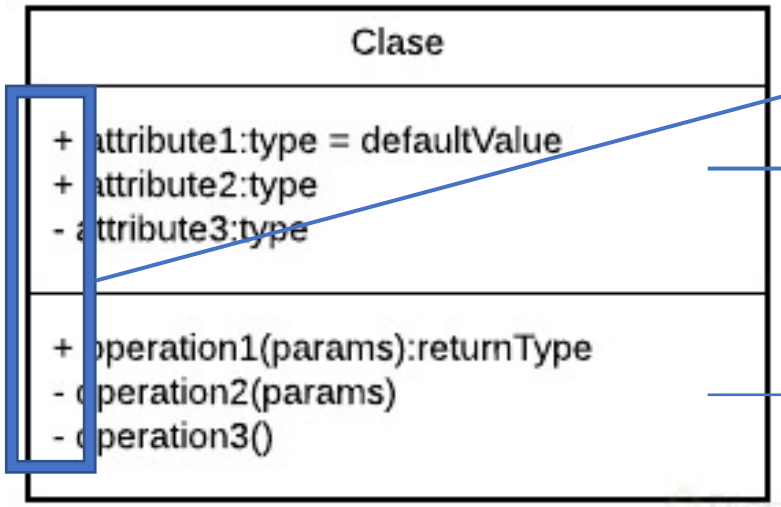
Por otro lado debemos saber que POO puede ser representado en un lenguaje estándar como UML

Una clase en UML puede ser representada:



Clases, Propiedades, Métodos y Objetos

La clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede mencionar que una clase es una plantilla genérica para un conjunto de objetos de similares características. Una clase define el estado y el comportamiento que todos los objetos creados a partir de esa clase tendrán.



Visibilidad del atributo o método (Encapsulamiento)

Atributos o propiedades

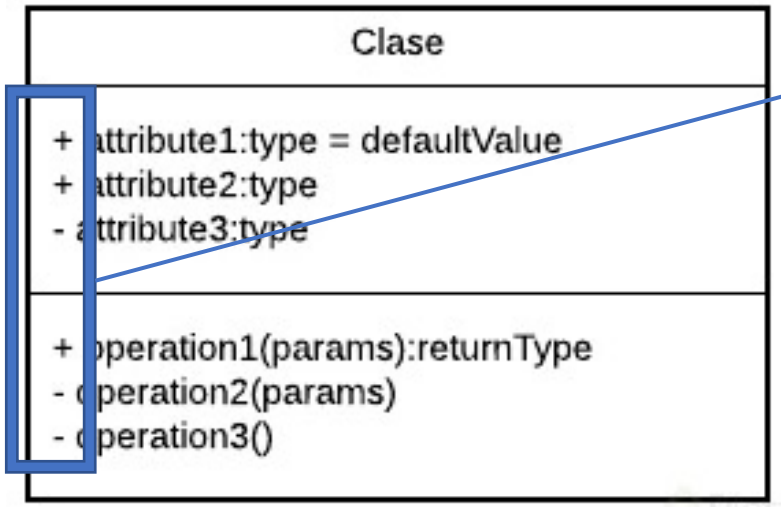
visibilidad nombre_atributo : tipo = valor-inicial { propiedades }

Operaciones o método de la clase

visibilidad nombre_funcion { parametros } : tipo-devuelto { propiedades }

Clases, Propiedades, Métodos y Objetos

Una clase en UML puede ser representada:

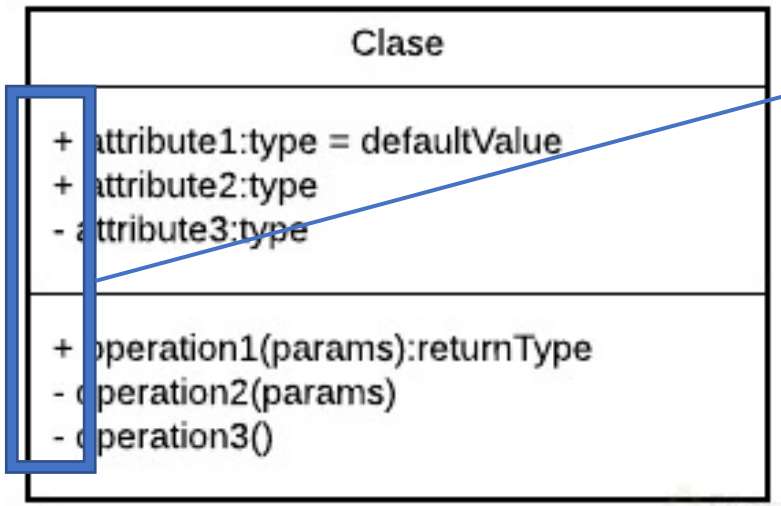


Visibilidad del atributo o método (Encapsulamiento)

- private
+ public
# protected

Clases, Propiedades, Métodos y Objetos

Una clase en UML puede ser representada:

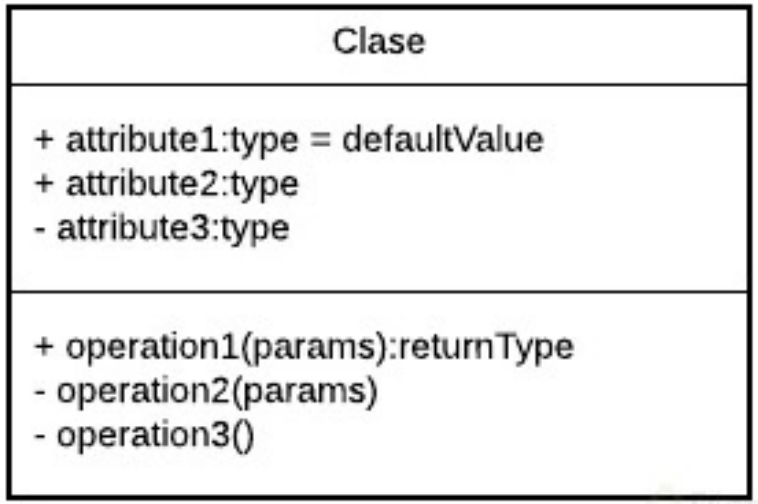


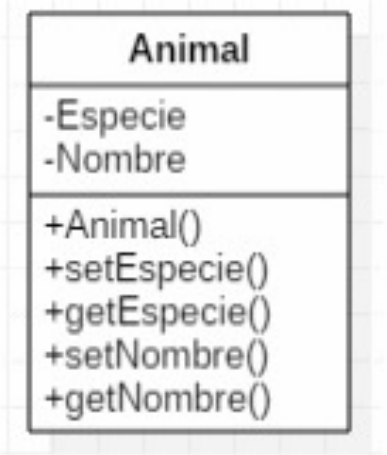
Visibilidad del atributo o método (Encapsulamiento)

- private: Se puede acceder al atributo o función únicamente desde la misma clase.
+ Public : Se puede acceder al atributo o función desde cualquier lugar de la aplicación.
# protected: el atributo o función puede ser accedida únicamente desde la misma clase o desde las clases que hereden de ella (clases derivadas).

Clases, Propiedades, Métodos y Objetos

Ejemplo





Ejemplo de una clase

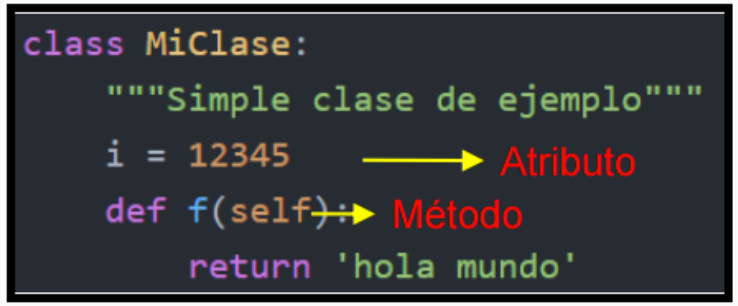
Clases, Propiedades, Métodos y Objetos

Atributo

Es una característica de un objeto, que ayuda a definir su estructura y permite diferenciarlo de otros objetos. Se define con un identificador y un tipo, el cual indica los valores que puede almacenar. El conjunto de valores de los campos definen el estado del objeto.

Métodos

Los métodos son los comportamientos o conductas de un objeto y permite identificar la forma en que actúa respecto a su entorno o respecto a otros objetos. Además, representa una operación o función que un objeto realiza. El conjunto de los métodos de un objeto determinan el comportamiento general del objeto.



Clases, Propiedades, Métodos y Objetos

Objeto

Los objetos de software, al igual que los objetos del mundo real, también tienen características y comportamientos. Un objeto de software mantiene sus características en uno o más atributos e implementa su comportamiento con métodos.

- Es una entidad real o abstracta, con un papel definido en el dominio del problema.
- Un objeto es una instancia de una clase, que tiene: identidad, estado(atributos) y comportamiento(metodos).

Para el ejemplo anterior de la Clase UniversityStudent (Estudiante universitario), la creacion de objetos se haria de la siguiente forma:

```python
class UniversityStudent:

    def __init__(self,id,name,gender,university,career,numsubjects):
        self.id = id
        self.name = name
        self.gender = gender
        self.university = university
        self.career = career
        self.numsubjects = numsubjects

    def inscribeSubjects(self):
        pass

    def cancelSubjects(self):
        pass

    def consultRatings(self):
        pass
```

Objeto

Ejemplo en Python

```python
student = UniversityStudent(123, "Pepe", "masculino", "UN", "Medicina", 8)
```

Parte Práctica

1) Todos los objetos deben pertenecer a una clase. De esa forma se van a poder diferenciar unos de otros, pues tendrán atributos y métodos específicos de esa clase y no de otra.
2) Para crear una clase se usa **class N**ombre_clase () -→ siempre el nobre de la clase empieza con mayúscula y luego debe terminar con los paréntesis y ":"

```
class Coche():
```

3) Luego se definen los atributos de la clase, recuerda identar (sangría), para especificar que estamos escribiendo dentro de la clase. En este caso el atributo es ruedas y se le da un valor = 4 .

```
class Coche():
        ruedas=4
```

4) Ahora vamos a agregar un método que en Python se hace de igual forma como se crea una función, con la palabra **def** seguido del nombre del método. Para diferenciar un método de una función se escribe (**self)**

```
class Coche():
        ruedas=4
        def desplazamiento(self):
                pass.
```
→ si aún no le asignamos nada al método debemos colocar pass

5) Ahora vamos a definir el método y sólo le colocaremos un print().

```
class Coche():
        ruedas=4
        def desplazamiento(self):
                print("El coche se esta desplazando sobre 4 ruedas")
```

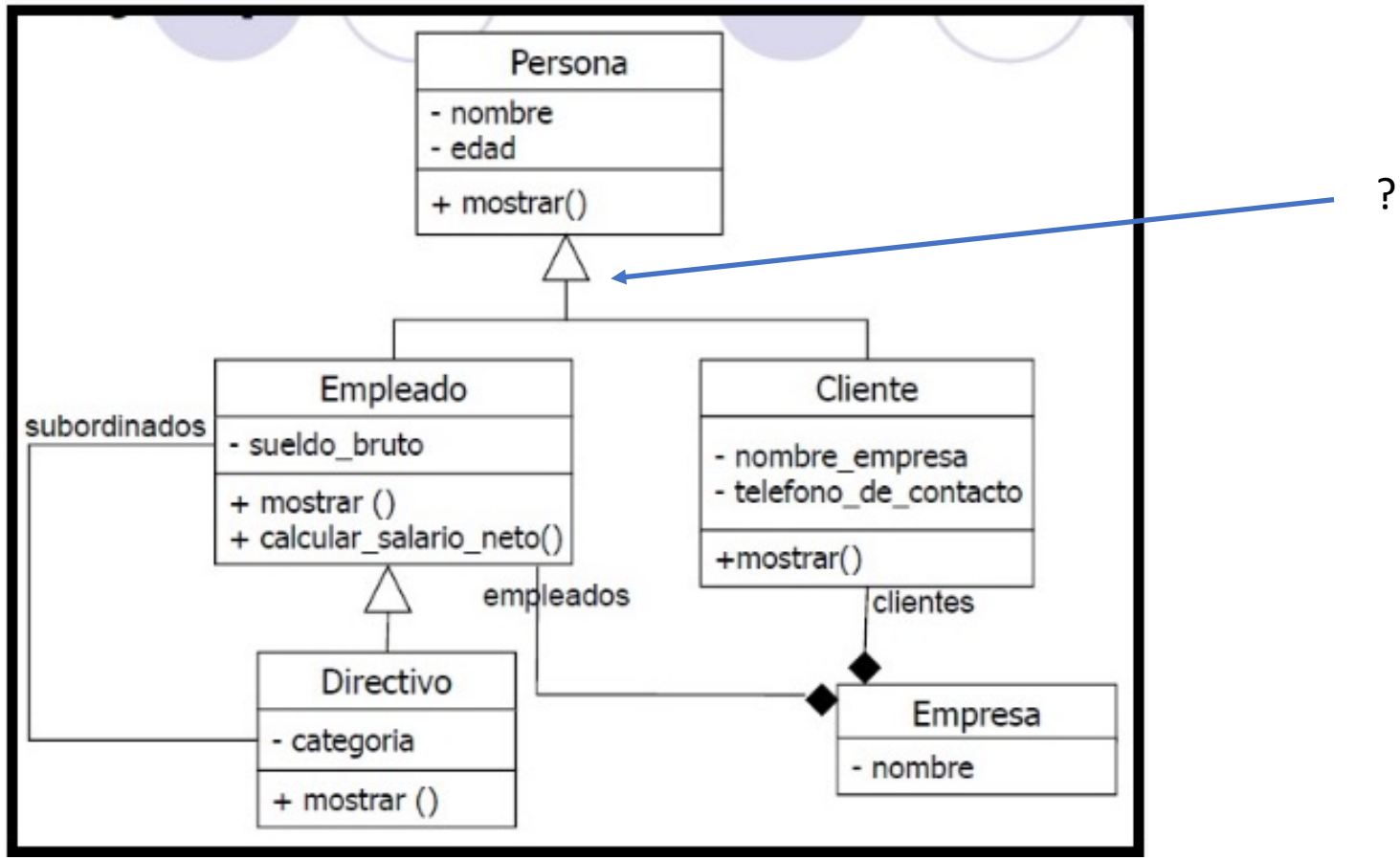
6) Ya tenemos una clase definida, por lo que vamos a crear objetos de esa clase. Creamos un objeto “miVehiculo” y después del “=” le estamos especificando a que clase pertenece el objeto que acabamos de crear.

```
miVehiculo=Coche()
```

Para una mayor comprensión de esto, ver jupyter-notebook sección A)

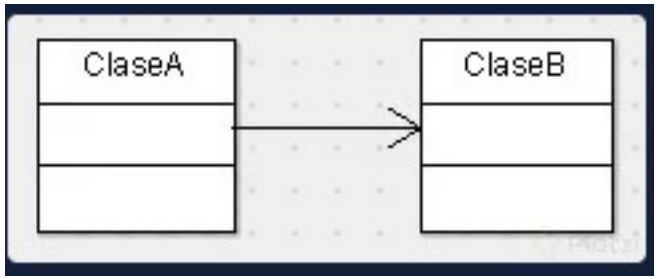
Diagramas de clases

Tipo de diagrama de estructura estática que representa la estructura de un sistema mostrando las clases, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.



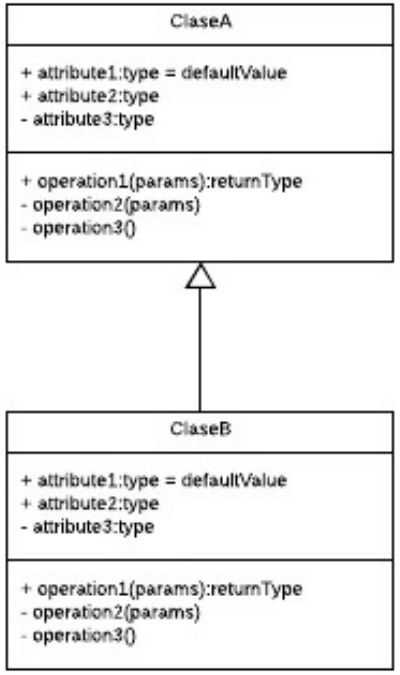
Herencia

Las relaciones que tendrá un elemento con otro se representan con flechas en UML y siempre que veamos este tipo de flecha se estará expresando la herencia.

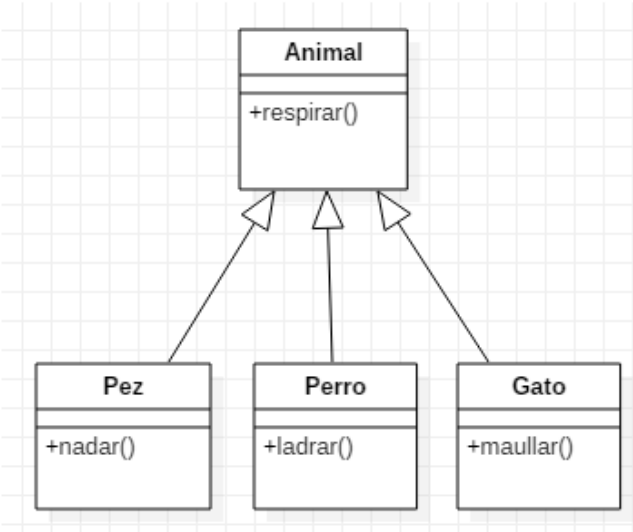


La ClaseA está asociada y depende de la ClaseB.
La clase A hereda de la clase B

La herencia se dibuja mediante un triángulo donde su eje superior apunta a la clase padre.



La clase B hereda de la clase A

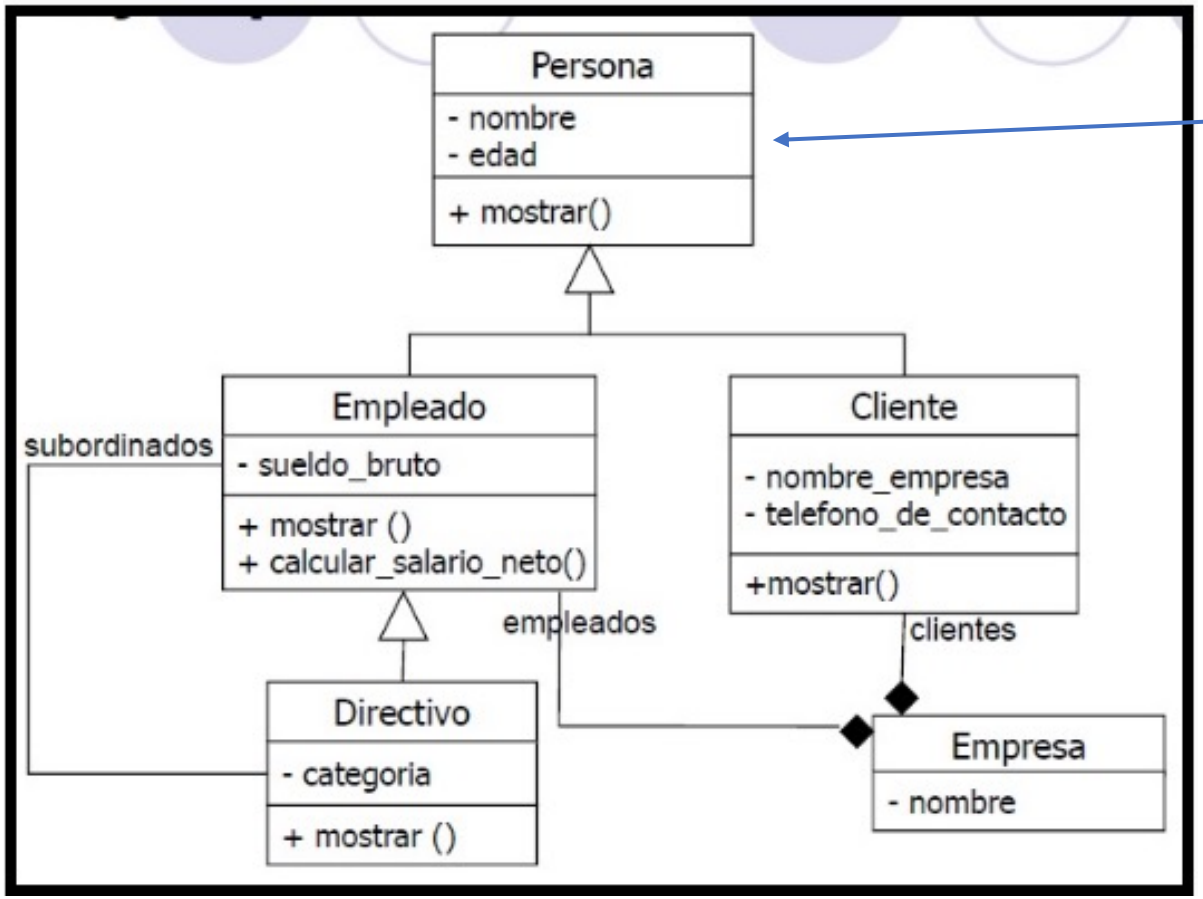


Ejemplo de herencia

Un pez, un perro y un gato son animales.

En este ejemplo, las tres clases (Pez, Perro, Gato) podrán utilizar la función respirar, ya que lo heredan de la clase animal, pero solamente la clase Pez podrá nadar, la clase Perro ladrar y la clase Gato maullar.

Herencia



La herencia se dibuja mediante un triángulo donde su eje superior apunta a la clase padre.

La visibilidad de los atributos

- **-** private
- **+** public
- **#** protected

La relación que tiene un objeto con otro se representa dependiendo de su categoría, **una relación de asociación es una línea sencilla** que **une las dos clases** vinculadas.

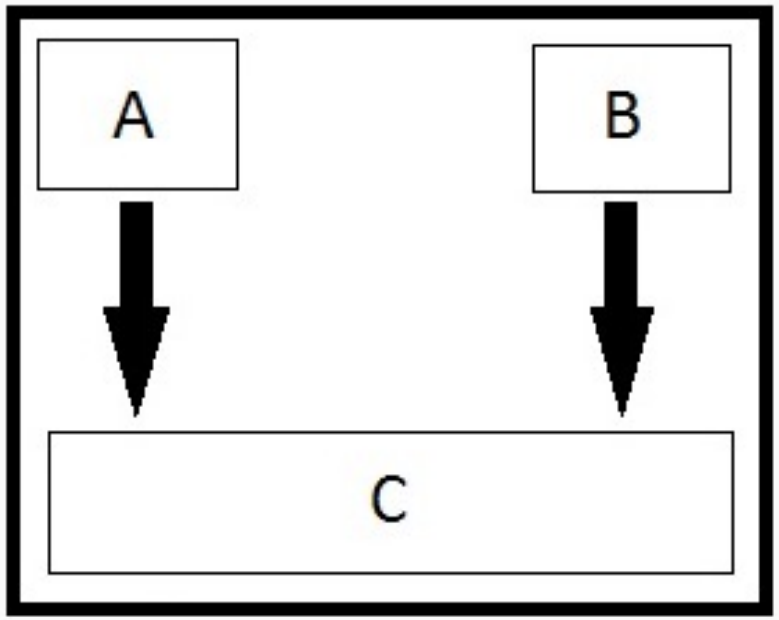
Una **relación de agregación** (existe una instancia de un objeto en una clase) se describe mediante **un rombo vacío.**

Una **relación de composición**(la existencia de una clase depende absolutamente de otra) se representa con **un rombo relleno.**

- **La clase *Empleado* y la clase *Cliente* heredan de la clase *Persona*.**
- **Todos los atributos de las clases son privados,**
- **Existe una relación de composición entre la clase *Empleado* y la clase *Empresa*, es decir, si un objeto *Empresa* deja de existir, la clase *Empleado* también va a dejar de existir.**

Herencia Múltiple

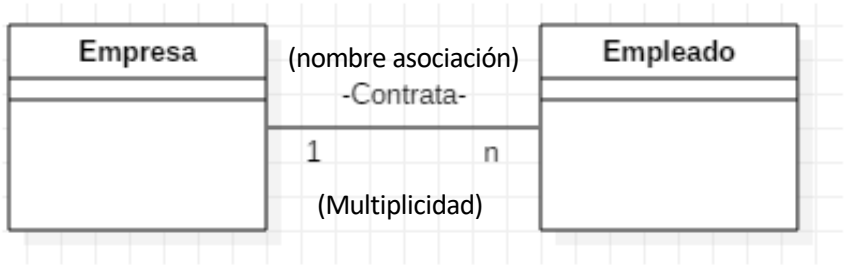
Es cuando una clase hereda de dos o más clases base.
La herencia permite reutilizar código ya permite a una clase heredar propiedades y comportamiento de otra clase.



La clase C hereda tanto de la clase A como de la clase B.

Para que podamos ver esto en código, te recomiendo ver jupyter-notebook B)

Relaciones



Ejemplo de relación Empresa-Empleado

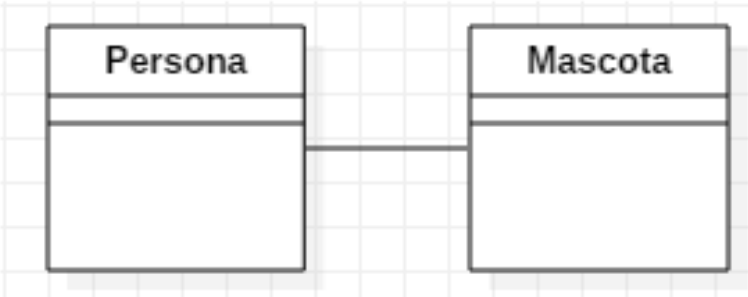
Asociación: se utiliza para representar dependencia semántica.

- Se representa con una simple línea continua que une las clases que están incluidas en la asociación.

Ej:

Un empleado es contratado por una empresa.

Una mascota pertenece a una persona



Ejemplo de asociación

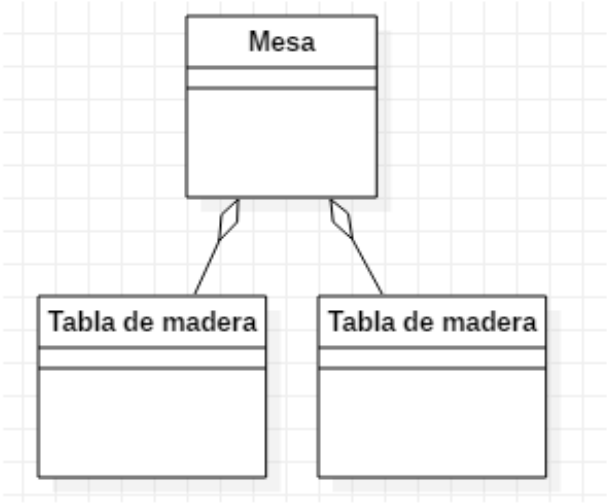
Relaciones

Agregación: Representación jerárquica que indica a un objeto y las partes que componen ese objeto. Representa relaciones en las que **un objeto es parte de otro**, pero aun así debe tener **existencia en sí mismo**.

- Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Ej:

Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa



Ejemplo de agregación

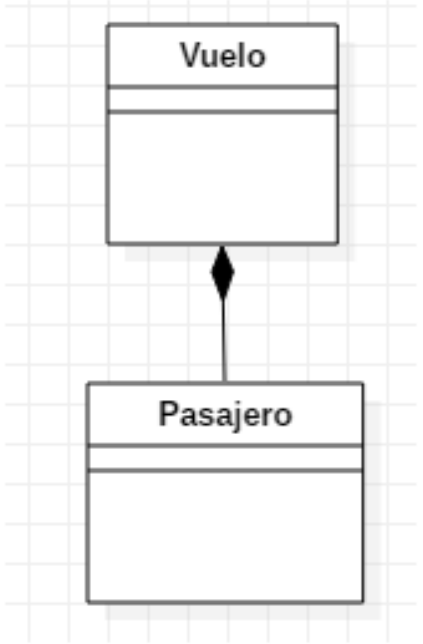
El tornillo podría formar parte de más objetos, por lo que interesa especialmente su **abstracción** en otra clase.

Relaciones

Composición: La composición es similar a la agregación, representa una **relación jerárquica entre un objeto y las partes que lo componen**, **pero de una forma más fuerte**.
Los elementos que forman parte no tienen sentido de existencia cuando el primero no existe. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos sino que dependen del elemento que componen. Además, suelen tener los mismos tiempo de vida. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.

- Se representa con una línea continua con un rombo relleno en la clase que es compuesta.

Ej:
Un vuelo de una compañía aérea está compuesto por pasajeros, que es lo mismo que decir que un pasajero está asignado a un vuelo



Ejemplo de composición

El tornillo podría formar parte de más objetos, por lo que interesa especialmente su **abstracción** en otra clase.

Diferencia entre agregación y composición
La diferencia entre agregación y composición es semántica, por lo que **a veces no está del todo definida**. Ninguna de las dos tienen análogos en muchos lenguajes de programación (como por ejemplo Java).