PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

AYUDANTÍA 1

Programación Orientada a Objetos

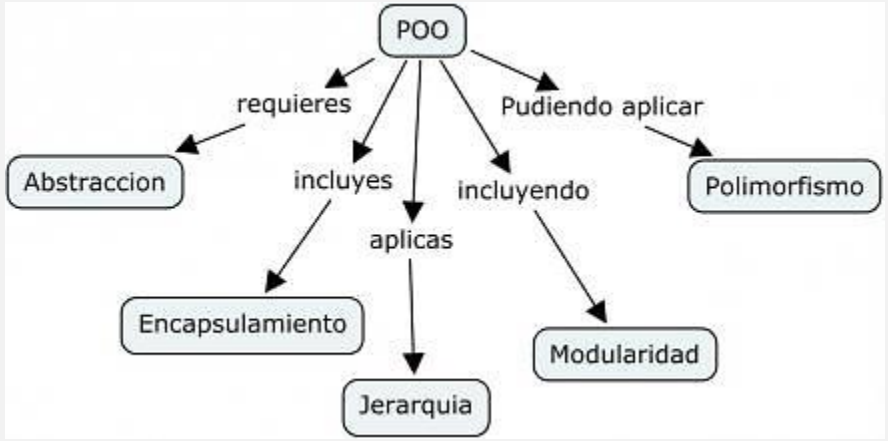
Ayudante: Felipe Fuentes Porras

¿POR QUÉ USARLO?

- Nos permite representar de forma virtual objetos del mundo real (o abstracciones).

- La forma en que OOP representa los objetos es una **abstracción, no necesitamos conocer como funciona en su totalidad el objeto.**

- Modelo simplificado de la realidad.



DEFINIR UNA CLASE EN PYTHON

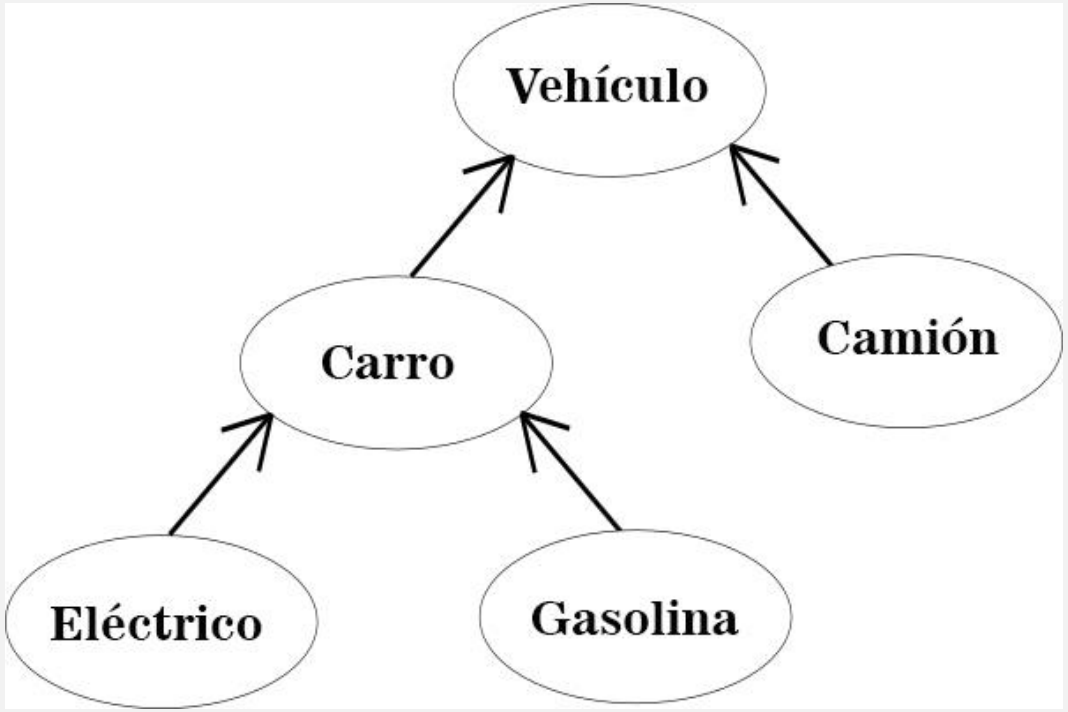
```python
class Departamento:
    '''Clase que representa un departamento en venta
       valor esta en UF.
    '''
    def __init__(self, _id, mts2, valor, num_dorms, num_banos):
        self._id = _id
        self.mts2 = mts2
        self.valor = valor
        self.num_dorms = num_dorms
        self.num_banos = num_banos
        self.vendido = False

    def vender(self):
        if not self.vendido:
            self.vendido = True
        else:
            print("Departamento {} ya se vendió".format(self._id))
```

```python
d1 = Departamento(_id=1, mts2=100, valor=5000, num_dorms=3, num_banos=2)
print(d1.vendido)
```

HERENCIA

- Le da estructura a la modelación.
- Permite ahorrar código y reutilizarlo.
- Facilita la modificación de funcionalidades.
- Permite añadir fácilmente nuevas funcionalidades a tu código.

EJEMPLO DE HERENCIA



HERENCIA EN PYTHON

```python
class Investigador:
    def __init__(self, area):
        self.area = area

class Docente:
    def __init__(self, departamento):
        self.departamento = departamento

class Academico(Docente, Investigador):
    def __init__(self, nombre, area_investigacion, departamento):
        #esto no es del todo correcto, coming soon...
        Investigador.__init__(self, area_investigacion)
        Docente.__init__(self, departamento)
        self.nombre = nombre
```

POLIMORFISMO

- Es cuando se envía un mensaje sintácticamente igual a distintos objetos, pero de tipos distintos.
- Se pueden modificar métodos de la clase y operadores.

```
def llamar(self):
    super().llamar()
    print("Llamando método en Subclase Derecha")
    self.num_llamadas_der += 1
```

```
def __repr__(self):
    return "\n".join("Producto: {} | Cantidad: {}".format(p, self.lista_productos[p]) for p in self.lista_productos.keys())
```

CLASES ABSTRACTAS

- Sirve para definir una “interfaz” o “esquema” que deben seguir todas sus clases hijas.
- Permite crear clases que no necesitan ni pueden ser instanciadas.

CLASE ABC PYTHON

```python
from abc import ABCMeta, abstractmethod

class Base(metaclass=ABCMeta):
    @abstractmethod
    def func_1(self):
        pass

    @abstractmethod
    def func_2(self):
        pass

class SubClase(Base):

    def func_1(self):
        pass

    def func_2(self):
        pass
```

ENCAPSULAMIENTO

- Se utiliza para “privatizar” métodos y atributos dentro de un objeto.
- Los métodos y atributos encapsulados solo pueden ser alterados y accedidos por operaciones definidas dentro del objeto.

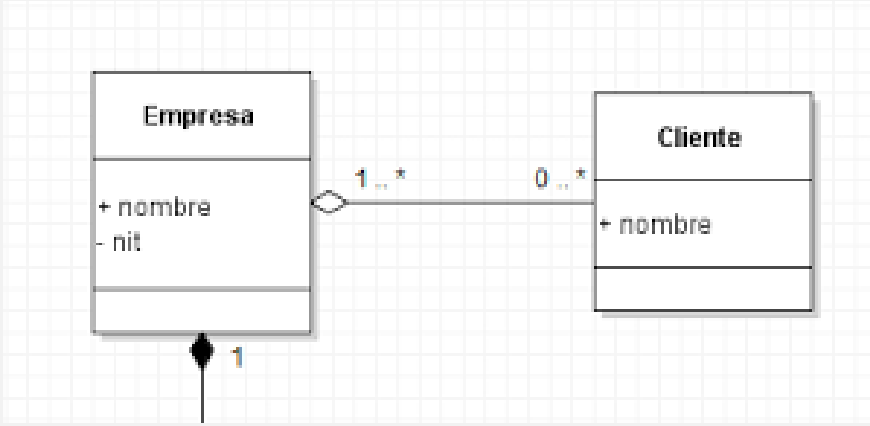
¡¡¡PROBLEMA!!!

- Python es un leguaje que no contiene una forma de privatizar realmente los atributos y métodos de un objeto.
- Pero existen alternativas:

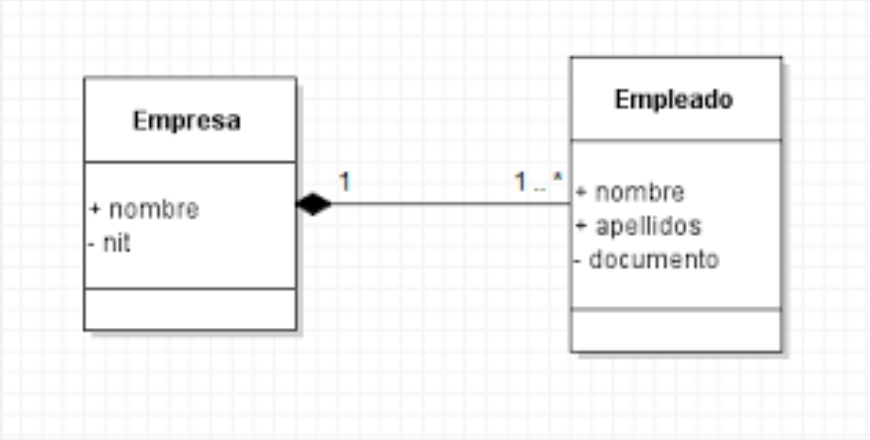
```
 9
10  class Student(object):
11      """docstring for Student"""
12      def __init__(self, arg):
13          super(Student, self).__init__()
14          self.__arg = arg
15
16      def setArg(self,newArg):
17          self.__arg = newArg
18
19      def getArg(self):
20          return self.__arg
21
22  p = Student(600)
```

AGREGACIÓN Y COMPOSICIÓN.

Agregación



Composición



DIFERENCIAS

	Agregación	**Composición**
Varias asociaciones comparten los componentes	Sí	No
Destrucción de los componentes al destruir el compuesto	No	Sí
Cardinalidad a nivel de compuesto	Cualquiera	0..1 ó 1
Representación	Rombo transparente	Rombo negro

EJERCICIO TALLER

Descripción del problema

El local de votación opera como un sistema con N filas de espera, en donde los votantes llegan aleatoriamente con una probabilidad p y se ubican en la fila correspondiente a su mesa. Un individuo vota cuando llega al principio de la fila, y el tiempo de votación (lo que demora en votar, sin considerar la espera) depende de la edad del votante. Considere finalmente que debido al contexto sanitario, el local de votación tiene un aforo máximo en su interior, por lo que cuando este es superado, las filas de cada mesa deben continuar en el exterior. El criterio para manejar las filas exterior y el momento en que un individuo pueden entrar al local debe ser definido por usted.

CONSIDERACIONES DEL EJERCICIO

El modelo a implementar debe considerar los siguientes elementos:

- Clases para todas las entidades relevantes a modelar.
- La jerarquía completa de clases para individuos, que considere una clase base abstracta (primera clase de la jerarquía) que solo defina la interfaz (atributos y métodos sin implementar).
- Dos clases que hereden de otra (puede no ser la misma).
- Clases que participen como atributos en otras (composición o agregación)
- Implementaciones del método `__str__()`, que entreguen información relevante del estado de los objetos.
- Dos sobrecargas de métodos (*override* - polimorfismo)

SIMULACIÓN

```
for t in range(t_max):
    if llega un auto nuevo

   if no hay vehículos en atención y sí hay vehículos en espera
       # verificar si hay cambios de cola
       # se atiende al primer vehículo de cada cola
   # se actualiza el tiempo de espera para la próxima atención
```

EJERCICIO 2

Introducción

La ONU les ha pedido ayuda para organizar una disputa a nivel mundial. Las diferentes ciudades del mundo comenzaron a discutir sobre cuál era la más popular y para poder encontrar una respuesta, la ONU decidió asignar a cada ciudad un color único. De esta manera, cada ciudad pintará los vehículos que lleguen a la ciudad de ese color, que luego, al ser contados, revelarán la ciudad más visitada y, por lo tanto, la más popular.

En este taller deberán modelar la situación expuesta anteriormente. Para esto, la ONU les envió un documento donde se explican los 3 entes principales que participan del problema: Personas, Vehículos y Ciudad.

EJERCICIO 2

Descripción

Para este problema, las ciudades se caracterizan por poseer un nombre, un color y la cantidad de vehículos pintados. Cada persona posee un nombre, una edad, un contador de horas conducidas, una lista con el nombre de las ciudades visitadas y un vehículo.

Por otro lado, los vehículos (en este caso autos y bicicletas) tienen un color, velocidad promedio, kilómetros recorridos, kilómetros máximos por recorrer, un contador de ciudades visitadas y un indicador de estado que puede ser: *'Funcionando'* o *'En Panne'*. Todos los vehículos son de color blanco, parten con 0 kilómetros recorridos y con el estado *'Funcionando'*. Los autos pueden andar hasta 1500 kilómetros y tienen una velocidad promedio de 80 kilómetros por hora. Por su parte, las bicicletas tienen como máximo 1000 kilómetros para recorrer y una velocidad promedio de 30 kilómetros por hora. Finalmente, todos los vehículos deben ser capaces de mostrar su eficiencia, la cual es calculada como la división entre las ciudades visitadas y el total de kilómetros recorridos.

EJERCICIO 2

Para identificar siempre la última ciudad visitada por cada persona, las ciudades **pintan** el vehículo de las personas que llegan a ellas con su color característico y el contador de vehículos pintados por las ciudades aumenta en 1 de forma correspondiente. A su vez, cada vez que un vehículo es pintado, su contador de ciudades visitadas aumenta en 1.

Un vehículo tiene la capacidad de **recorrer** una cantidad de kilómetros (1500 o 1000). Al momento de realizar tal acción (avanzar/recorrer), el contador de kilómetros recorridos del vehículo aumenta y si

EJERCICIO 2

este llega a superar los kilómetros máximos posibles, el estado del vehículo pasará a ser *'En Panne'* y solo recorrerá los kilómetros necesarios para que la cantidad de kilómetros recorridos sea igual a su capacidad máxima. Cada vez que a un vehículo se le pide recorrer una distancia, este retorna el tiempo que se demoró según la velocidad promedio que posee.

Por último, una persona puede **viajar** siempre y cuando se le indique la ciudad y los kilómetros necesarios para llegar a esa ciudad. Al momento que una persona decida viajar, el vehículo deberá recorrer la distancia indicada y el contador de horas conduciendo de la persona será aumentado según el tiempo que le tomó al vehículo recorrer la distancia. En caso de que el vehículo llegue a su destino, en consola se deberá informar el siguiente mensaje: 'La persona *nombre* llegó a *nombre_ciudad* en *tiempo_demorado* horas y ahora su vehículo es de color *color_vehiculo*'. En caso contrario deberá mostrar el siguiente mensaje: 'La persona *nombre* no logró llegar a *nombre_ciudad*, se quedó en panne a las *tiempo_demorado* horas de viaje.'[1]

EJERCICIO 2

- Vehículos: el tipo del vehículo, su color actual del vehículo, los kilómetros recorridos y el estado actual.

  `Bicicleta: color actual azul, ha recorrido 34 kilómetros, estado es Funcionando`

- Personas: el nombre de ella, su edad, la cantidad de horas, las ciudades visitadas y el tipo y color de su vehículo. Ejemplo:

  `Juanito Perez: edad 28 años, su vehículo es bicicleta de color rojo, ha conducido 128 horas.`
  `Ciudades Visitadas: París, Santiago, Lima`

- Ciudad: el nombre de ella, su color y la cantidad de vehículos pintados