UNIVERSIDAD TECNOLOGICA NACIONAL

POO: Programación orientada a objetos (Parte 3)

Programación 2

Estado inicial / Constructor

```
class Auto():
    largoChasis=250
    anchoChasis=120
    ruedas=4
    enmarcha=False

    def arrancar(self):
      (...)

    def estado(self):
      (...)

miAuto=Auto()
miAuto2=Auto()
miAuto3=Auto()
```

Estado inicial / Constructor

```
class Auto():

    def __init__(self):
        self. largoChasis=250
        self.anchoChasis=120
        self.ruedas=4
        self.enmarcha=False

    def arrancar(self):
        (...)

    def estado(self):
        (...)

miAuto=Auto()
miAuto2=Auto()
miAuto3=Auto()
```

Encapsulamiento

```
class Auto():

    def __init__(self):
        self. largoChasis=250
        self.anchoChasis=120
        self.ruedas=4
        self.enmarcha=False

    def arrancar(self):
        (...)

    def estado(self):
        (...)

miAuto=Auto()
miAuto2=Auto()

miAuto2.ruedas = 2              #Esto no debería poder permitirse
```

Encapsulación

```
class Auto():

    def __init__(self):
        self. largoChasis=250
        self.anchoChasis=120
        self.__ruedas=4
        self.enmarcha=False

    def arrancar(self):
        (...)

    def estado(self):
        (...)

miAuto=Auto()
miAuto2=Auto()
```

Atributo o propiedad encapsulada

Encapsulación

```
class Auto():

    def __init__(self):
        self. largoChasis=250
        self.anchoChasis=120
        self.__ruedas=4
        self.enmarcha=False

    def arrancar(self):
        self.chequeo_interno()
        (...)

    def estado(self):
        (...)

    def chequeo_interno(self):
        (...)

miAuto=Auto()
miAuto2=Auto()
```

Antes de arrancar se hace el chequeo interno

Nuevo método que se encarga de hacer un chequeo interno antes de arrancar

Encapsulación

```
class Auto():

    def __init__(self):
        self. largoChasis=250
        self.anchoChasis=120
        self.__ruedas=4
        self.enmarcha=False

    def arrancar(self):
        self.__chequeo_interno()
        (...)

    def estado(self):
        (...)

    def __chequeo_interno(self):
        (...)

miAuto=Auto()
miAuto2=Auto()
```

Encapsulamos el método

Encapsulación, cuándo encapsular?

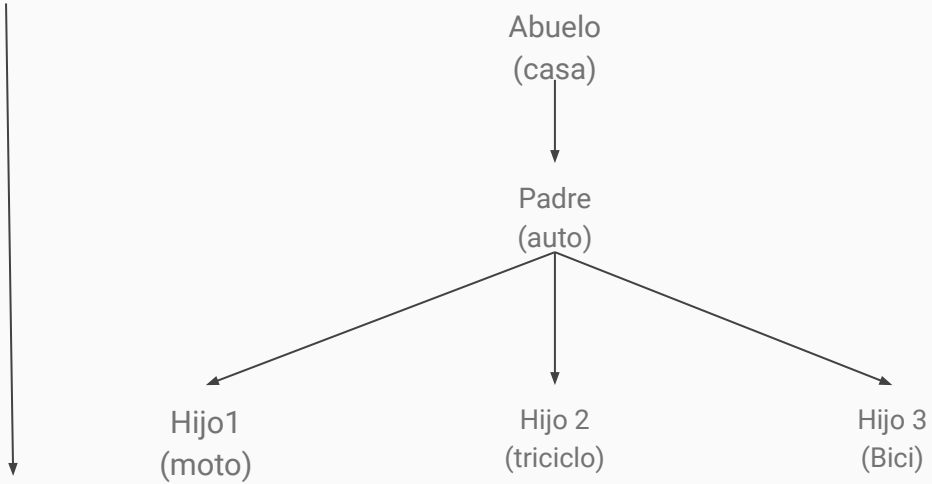
```
class Auto():

    def __init__(self):
        self. largoChasis=250
        self.anchoChasis=120
        self.__ruedas=4
        self.enmarcha=False

    def arrancar(self):
        self.__chequeo_interno()
        (...)

    def estado(self):
        (...)

    def __chequeo_interno(self):
        (...)

miAuto=Auto()
miAuto2=Auto()
```

No existe una regla fija para encapsular.

Se deben encapsular variables o se deben encapsular métodos cuando la clase así lo necesite y eso depende del comportamiento que quieras que tenga esa clase según el criterio como programador o según los requerimientos específicos del sistema

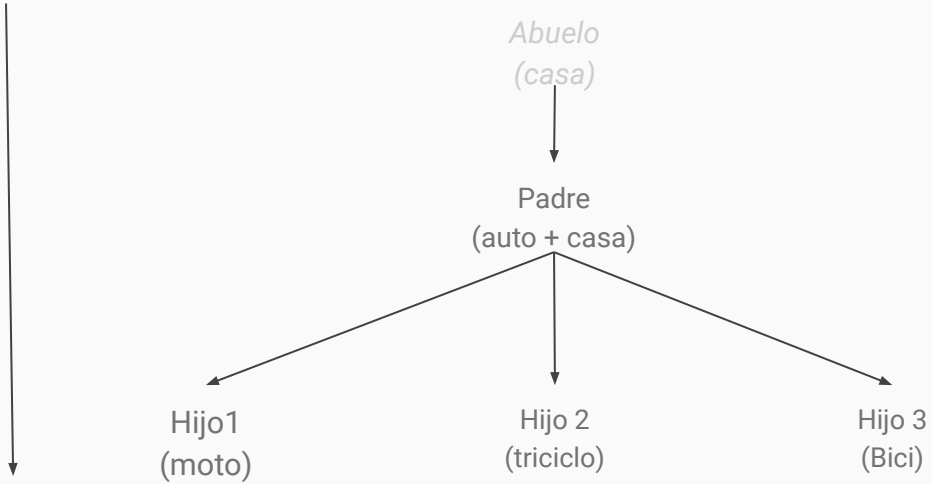
Herencia

Jerarquía de herencia

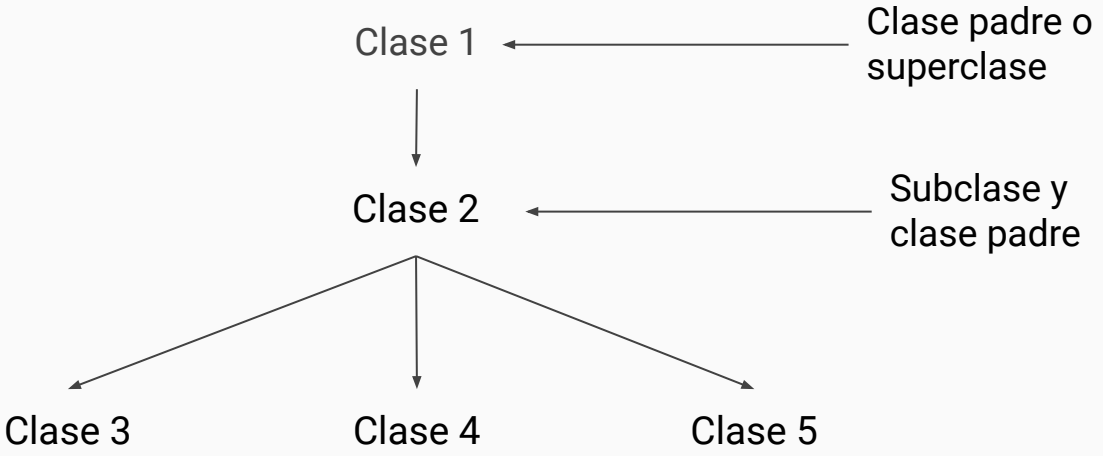


Herencia

Jerarquía de herencia



Herencia


Clase 1
Clase padre o superclase
Clase 2
Subclase y clase padre
Clase 3
Clase 4
Clase 5

Herencia, ¿Para qué sirve?

Para reutilizar código en caso de crear objetos similares.

- Auto
- Camión
- Moto
- Colectivo

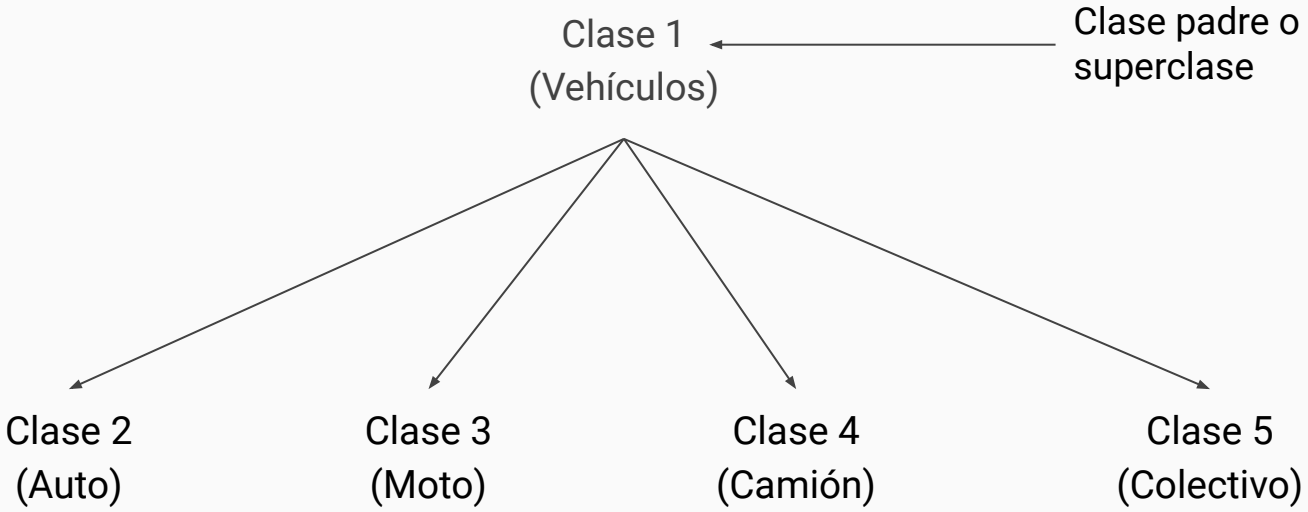
¿Qué característica en común tienen todos los objetos que hay que crear?

MARCA
MODELO

¿Qué comportamientos en común tienen todos los objetos?

ARRANCAN
ACELERAN
FRENAN

Herencia


Clase 1
(Vehículos)
Clase padre o
superclase
Clase 2
(Auto)
Clase 3
(Moto)
Clase 4
(Camión)
Clase 5
(Colectivo)