

Polimorfismo y EDD 101

Ayudantía 1

¿Cómo decidir si algo es clase
o no?

La asociación solo cuida perros y gatos, aunque en un futuro les gustaría agregar otros animales como los cobayos y erizos. Cada uno de los animales tiene un nombre, un color principal y sexo (macho o hembra). Los animales pueden jugar y comer. Los perros pueden ladrar y los gatos pueden maullar.

La asociación solo cuida perros y gatos, aunque en un futuro les gustaría agregar otros animales como los cobayos y erizos.

Cada uno de los animales tiene un nombre, un color principal y sexo (macho o hembra). **Los animales pueden jugar y comer.**

Los **perros pueden ladrar** y los **gatos pueden maullar.**

Los animales tienen distintas personalidades. Estas definen las horas de juego individual, horas de juego grupal, horas de sueño, horas de regaloneo y cantidad de comidas al día. Hasta ahora los miembros de la organización han identificado dos tipos de personalidades: juguetón y egoísta.

Además de estos valores en los parámetros las personalidades pueden cambiar la expresión de las acciones jugar y comer.

Los animales tienen distintas personalidades. Estas definen las horas de juego individual, horas de juego grupal, horas de sueño, horas de regaloneo y cantidad de comidas al día. Hasta ahora los miembros de la organización han identificado dos tipos de personalidades: juguetón y egoísta.

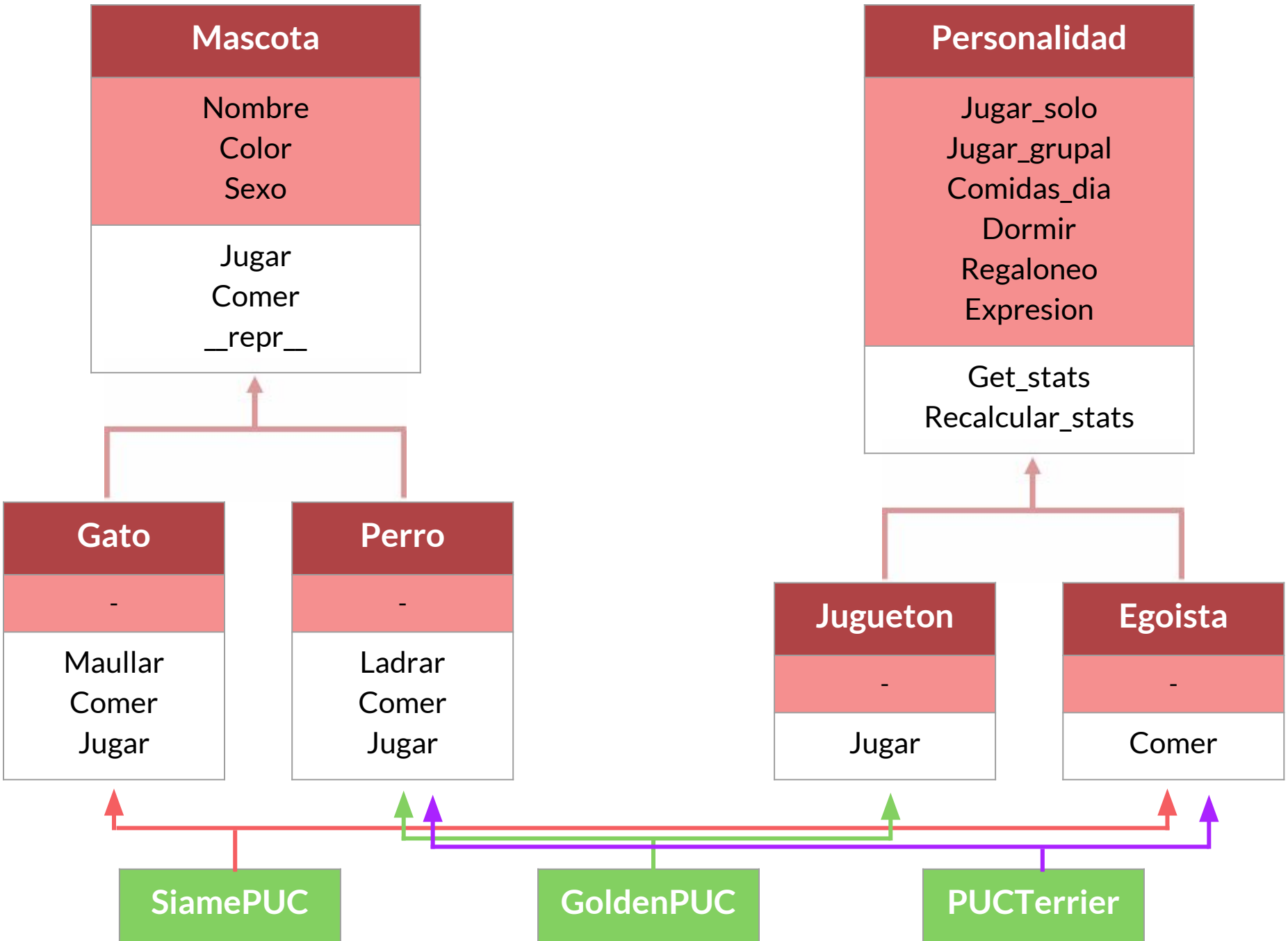
Además de estos valores en los parámetros las personalidades **pueden cambiar la expresión de las acciones jugar y comer.**

Los voluntarios se han dado cuenta de que hay dos razas de perros y una de gatos:

- GoldenPUC: es un perro de personalidad juguetona. En estos perros el parámetro de expresión de personalidad se multiplica por 1,1 para los machos y por 0,9 para las hembras.
- PUCTerrier: ...
- SiamePUC: ...

Los voluntarios se han dado cuenta de que hay dos razas de perros y una de gatos:

- GoldenPUC: **es un perro de personalidad juguetona.** En estos perros el parámetro de expresión de personalidad se multiplica por 1,1 para los machos y por 0,9 para las hembras.
- PUCTerrier: ...
- SiamePUC: ...



¿Cómo pasamos este modelo
a código?

¿Cómo pasamos este modelo a código?

Utilizamos:

- `*argv`
- `**kwargs`
- Clases y herencia
- `super.__init__()`
- Sobrescribir métodos

***argv**

Non-keyworded variable length argument tuple

Cuando se utiliza?

- Enviar cantidad variable de argumentos a una función
- Argumentos en orden y sin keyword (lista)

```
def test(*argv):  
    for arg in argv:  
        print("siguiente argumento de *argv : {}".format(arg))  
  
test('hola', 'como', 'va', 'todo')  
test('hola', 'como', 'va')  
test('hola', 'como')  
test('hola')  
test()
```

Todos Funcionan!

****kwargs**

Keyworded variable-length argument list

Cuando se utiliza?

- Enviar cantidad variable de argumentos a una función
- Argumentos no ordenados pero con keyword (diccionario)

super().__init__()

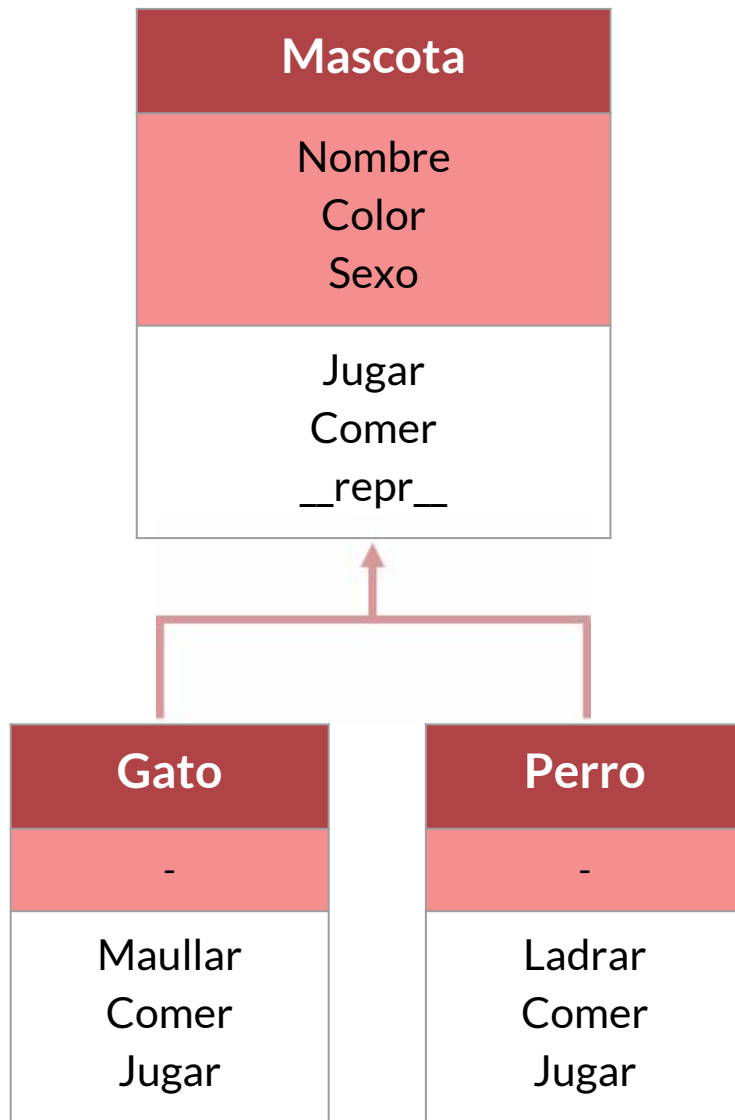
```
class Contacto:
```

```
    def __init__(self, nombre, email):  
        self.nombre = nombre  
        self.email = email
```

```
class Cliente(Contacto):
```

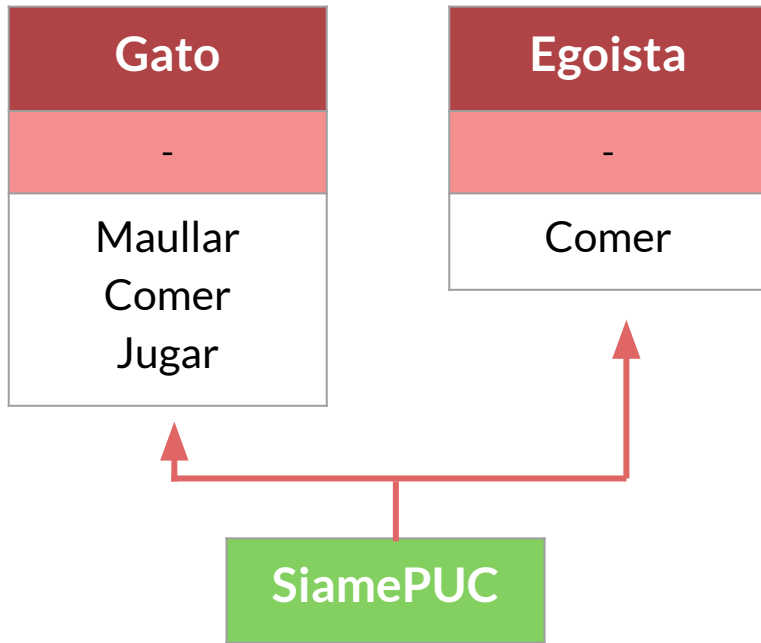
```
    def __init__(self, telefono, nombre, email):  
        super().__init__(nombre, email)  
        self.telefono = telefono
```

¿Cómo sobrescribir
métodos?



Gato y Perro son subclase de Mascota

¿Cómo sé si un objeto es instancia de alguno de ellos?



```
gato = SiamePUC()  
gato2 = SiamePUC()  
gato3 = SiamePUC()
```

Cada uno (gato1, gato2 y gato3) es una instancia de mi clase SiamePUC

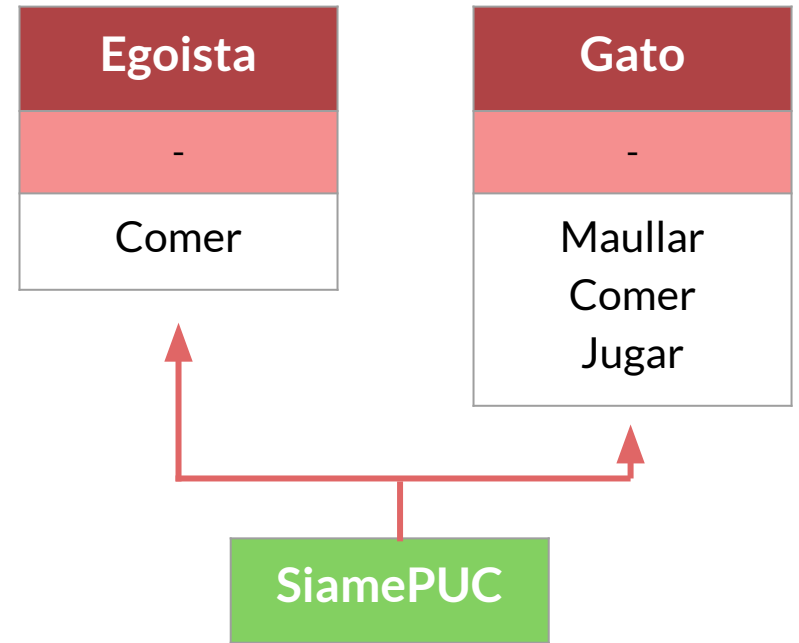
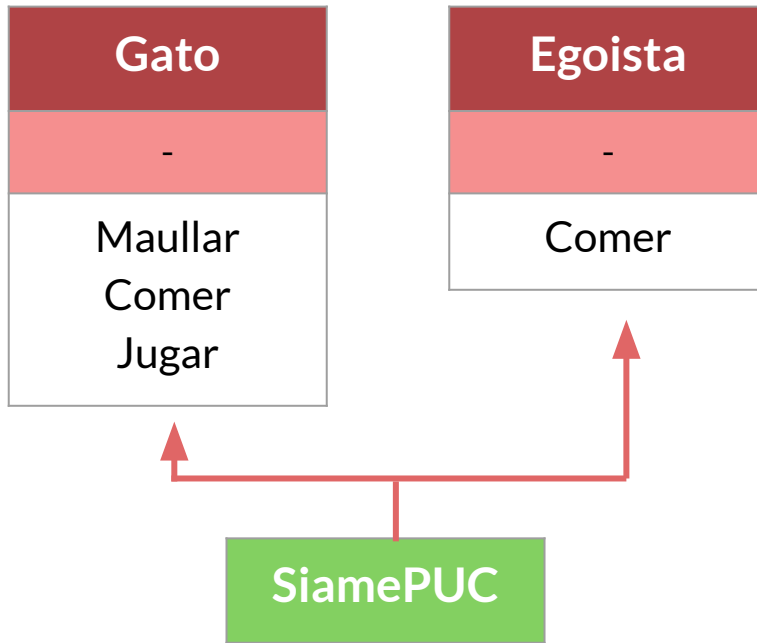
```
isinstance(gato, SiamePUC)  
isinstance(gato, Gato)
```

La personalidad egoísta hace que la acción comer imprima lo siguiente:

1. “Quiero comida”
2. el mensaje de la acción comer original según la especie
3. el mensaje de ladrar o maullar, según la especie

```
class Egoista(Personalidad):  
  
    ...  
  
    def comer(self):  
        print("Quiero comida")  
  
        if isinstance(self, Perro):  
            super().comer()  
            self.ladRAR()  
        else:  
            super().comer()  
            self.mauLLar()
```

¿Es lo mismo?



Estructuras de Datos

Listas

```
lista = [arg1, arg2, arg3, ..., argn]
```

Ejemplo:

```
cosas = ['mochila', 7, ['gato', 'animal']]
```

- Puede contener distintos tipos de datos
- Son una estructura ordenada
- Son estructuras **mutables**
- Los puedes recorrer con un for

Listas

`lista = list() = [] = [a, b, c] → crea la lista`

`lista.append(4) → Agrega un argumento al final de la lista`

`lista.sort(reverse = True) → Retorna la lista ordenada`

`lista[a:b:c] → Recorre la lista desde a hasta b (sin incluir) con pasos de largo c`

`for i in lista: → se itera sobre los elementos de la lista en orden`

Tuplas

```
tupla = (arg1, arg2, arg3, ..., argn)
```

Ejemplo:

```
cosas = ('mochila', 7, ['gato', 'animal'])
```

- Puede contener distintos tipos de datos
- Son estructuras **inmutables**
- Se recorren igual que una lista!

NamedTuples

```
tupla = namedtuple(Nombre, 'attr1, attr2, ..., attrn')
```

Ejemplo:

```
Curso = namedtuple('Ramo', 'Nombre credits departamento')  
c1 = Ramo('Programacion avanzada', 10, 'DCC')
```

- Se define una tupla con el nombre de sus atributos
- Se puede usar en un return
- ¡Se debe importar de collections!

Conjuntos

Es una colección de **elementos únicos desordenados**

- Encontrar los items de `lista1` que están en `lista2`

```
lista1 = [1, 2, 3, 3]
```

```
lista2 = [1, 4, 5]
```

```
set(lista1).intersection(set(lista2))
```

- Encontrar los elementos únicos no repetidos

```
set(lista1)
```

Diccionarios

```
dict = {'key1': 'value1', 'key2': 'value2', 'key3':  
        'value3'}
```

Ejemplo:

```
ayudante = {'nombre': 'Anders', 'Major': 'Computacion',  
            'jerarquia': 'TPD'}
```

- Crear relaciones de atributo-valor correspondiente
- No importa el orden
- Estructura no posicional → valores siempre se buscan por su llave

Diccionarios: métodos

`del dict['key1']` → elimina llave con su atributo

`dict['key1'] = valor_x` → modifica/crea valor correspondiente a key1 con valor_x

`dict.keys()` → retorna todas las llaves del diccionario

`dict.values()` → retorna todos los valores del diccionario

`for i in dict:` → se itera sobre las llaves del diccionario

Ojo: si solo se itera sobre un diccionario se podría haber utilizado una lista, lo que es una mejor práctica.