Solución taller 5 Python.

Programación orientada a objetos. Grupo 1

Alejandro Pérez Barrera, alperezba@unal.edu.co, C.C. 1023629729

1. Se puede implementar un método ruido separado en la clase Pájaro:

```
25
26      def ruido(self):
27          return "cantar y silbar"
28
```

De esta manera si se llama el método para un pájaro

```
1   from Pajaro import Pajaro
2   if __name__=="__main__":
3       ave=Pajaro("Gabriel",7,"Canario","Amarillo")
4
5       print(ave.ruido())
6
```

Se obtiene un mensaje diferente al predeterminado de Animal

```
13/python.exe "c:/
cantar y silbar
```

2. Esta clase adoptará el constructor predeterminado de Python (El cual no hace nada), y heredará los métodos y atributos de la superclase.

3. Como los atributos de la clase Ser Vivo son privados, no se heredan, en su lugar, sería necesario añadir la palabra super() para que estos atributos puedan ser iguales.

```
1   from SerVivo import SerVivo
2
3   class Animal(SerVivo):
4       _totalCreados = 0
5
6       def __init__ (self, nombre, edad, raza):
7           super().__init__(nombre,edad) #Reemplazo
8           #self._nombre = nombre
9           #self._edad = edad
10          self._raza = raza
11          Animal._totalCreados += 1
```

4. Es posible utilizar super()

```
    def setNombre(self, nombre):
        super().setNombre(nombre)

    def getNombre(self):
        return super().getNombre()
```

5. No es heredado, porque Animal crea su propio atributo con el mismo nombre, **ocultando** al de su superclase. Por lo que sí, ocurre ocultamiento.

6. Si, porque comparten el mismo nombre, y sobre escriben al método de la clase superior:

```
from SerVivo import SerVivo
from Animal import Animal
from Pajaro import Pajaro
from Gato import Gato
if __name__=="__main__":
    ave=Pajaro("Gabriel",7,"Canario","Amarillo") #Se crea un Pájaro

    animal=Animal("Noko", 3,"Ciervo") #Se crea un Animal


    print(f"Pájaros: {Pajaro.getTotalCreados()}") #Hay un pájaro en total
    print(f"Animales: {Animal.getTotalCreados()}") #Hay dos animales en total (Animal + Pájaro)
    print(f"Seres Vivos: {SerVivo.getTotalCreados()}") #SerVivo no lleva ninguna cuenta de fábrica

```

Animal y Pájaro van a hacer su propia cuenta, pero como Animal y Ser Vivo no comparten inicializador, este no cuenta ningún animal, y llevan valores distintos, los cuales son almacenados y leídos en/desde espacios de memoria diferentes.

```
Pájaros: 1
Animales: 2
Seres Vivos: 0
```

7. Una clase que herede de Persona estaría heredando *__init__()*, *aduenarAnimal()* y *getTotalCreados()*, por parte de Persona; y *setNombre()*, *getNombre()*, *setEdad()*, *getEdad()* y *getTotalCreados()* de la clase Ser Vivo. Esto es, si la subclase no sobre escribe algún método.

8. Al método *aduenarAnimal()* solamente se le pueden pasar objetos que tengan el método *ruido()*, es decir, objetos de la clase Animal. Y si, se le pueden pasar objetos de tipo Ser Vivo, siempre y cuando estos sean simultáneamente de tipo animal, una vez más, debido al requisito del método *ruido()*.

9. Primero es necesario “conectar” a Animal con Ser Vivo por medio del constructor:

```python
from SerVivo import SerVivo

class Animal(SerVivo):
    _totalCreados = 0

    def __init__ (self, nombre, edad, raza):
        super().__init__(nombre,edad) #Ahora se llama a SerVivo
        #self._nombre = nombre
        #self._edad = edad
        self._raza = raza
        Animal._totalCreados += 1
```

De esta manera el contador de Ser Vivo puede hacer su trabajo

```python
class SerVivo:
    _totalCreados = 0

    def __init__ (self, nombre, edad):
        self._nombre = nombre
        self._edad = edad
        SerVivo._totalCreados += 1
```

Y si se crean seres vivos

```python
from SerVivo import SerVivo
from Animal import Animal
from Pajaro import Pajaro
from Perro import Perro
from Persona import Persona
if __name__=="__main__":
    ave=Pajaro("Gabriel",7,"Canario","Amarillo") #Se crea un Pájaro
    perro=Perro("Inu", 12,"Criollo","Oscuro") #Se crea un Perro
    humano=Persona("Franco",22)#Se crea un humano


    print(f"Pájaros: {Pajaro.getTotalCreados()}") #Hay un pájaro en total
    print(f"Perros: {Perro.getTotalCreados()}") #Hay un perro en total
    print(f"Personas: {Persona.getTotalCreados()}") #Hay una persona en total
    print(f"Animales: {Animal.getTotalCreados()}") #Hay dos animales en total (Perro + Pájaro)
    print(f"Seres Vivos: {SerVivo.getTotalCreados()}") #Hay tres animales en total (Perro + Pájaro + Persona)

```

Se lleva la cuenta de cada tipo

```
Pájaros: 1
Perros: 1
Personas: 1
Animales: 2
Seres Vivos: 3
```

10 . En Python no existe la sobrecarga, por lo que el método es sobre escrito en la clase Perro,

```
18      print(perro.getRaza())
19
```

donde si no se incluye el atributo de *tipo*, se genera un error

```
    print(perro.getRaza())
          ^^^^^^^^^^^^^^^^^
TypeError: Perro.getRaza() missing 1 required positional argument: 'tipo'
```

el cual no ocurre con otras clases

```
print(ave.getRaza())
```

→→→

```
Canario
```

sin embargo, si se incluye el atributo *tipo* al llamar al método desde un objeto tipo perro, el código se ejecuta sin problemas:

```
17
18      print(perro.getRaza(", Arroz con pollo"))
19
```

```
Criollo, Arroz con pollo
```